

NILO CÉSAR REICHEMBACH

**UTILIZAÇÃO DE UMA LINGUAGEM DE
PROGRAMAÇÃO QUÂNTICA NO ESTUDO
DO ALGORITMO DE GROVER**

REALEZA

2015

NILO CÉSAR REICHEMBACH

**UTILIZAÇÃO DE UMA LINGUAGEM DE
PROGRAMAÇÃO QUÂNTICA NO ESTUDO DO
ALGORITMO DE GROVER**

Trabalho de conclusão de curso apresentado
como requisito para a obtenção do grau de
licenciado em física da Universidade Federal
da Fronteira Sul.

**UNIVERSIDADE FEDERAL DA FRONTEIRA SUL
FÍSICA-LICENCIATURA**

Orientador: Eduardo de Almeida

REALEZA

2015

NILO CÉSAR REICHEMBACH

UTILIZAÇÃO DE UMA LINGUAGEM DE PROGRAMAÇÃO QUÂNTICA NO ESTUDO
DO ALGORITMO DE GROVER/ **NILO CÉSAR REICHEMBACH. – REALEZA, 2015**
85 p.

Orientador: Eduardo de Almeida

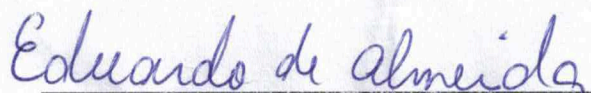
TRABALHO DE CONCLUSÃO DE CURSO –
UNIVERSIDADE FEDERAL DA FRONTEIRA SUL
FÍSICA-LICENCIATURA, 2015.

NILO CÉSAR REICHEMBACH

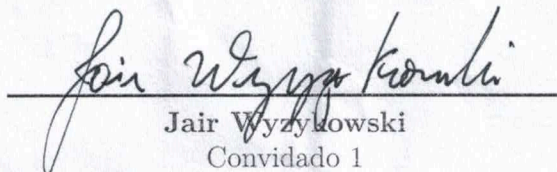
UTILIZAÇÃO DE UMA LINGUAGEM DE PROGRAMAÇÃO QUÂNTICA NO ESTUDO DO ALGORITMO DE GROVER

Trabalho de conclusão de curso apresentado
como requisito para a obtenção do grau de
licenciado em física da Universidade Federal
da Fronteira Sul.

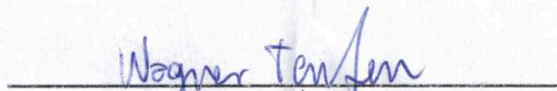
Trabalho aprovado. REALEZA, 09 Dezembro de 2015:



Eduardo de Almeida
Orientador



Jair Wyzykowski
Convidado 1



Wagner Tenfen
Convidado 2

REALEZA
2015

Agradecimentos

O agradecimento especial é dedicado aos meus pais Rodolfo e Balvina e também a minha noiva Fernanda. Devido aos anos de apoio prestado.

Os agradecimentos especiais são direcionados ao professor orientador deste trabalho, Eduardo de Almeida. Pela ajuda e paciência nas orientações.

Lista de ilustrações

Figura 1 – Sistema como a linguagem híbrida.	37
Figura 2 – Exemplo de script do QCL	38
Figura 3 – Gráfico da Lei de Moore (PINTO et al., 2015)	46
Figura 4 – Representação de um q-bit na esfera de Bloch	48
Figura 5 – Aplicação do algoritmo de Grover	56
Figura 6 – Circuito para iteração de Grover	57
Figura 7 – Representação geométrica a implementação G	58

Lista de tabelas

Tabela 1	– Ação computacional de XOR	31
Tabela 2	– Definição das variáveis	38
Tabela 3	– Tabela com demais funções importantes em QCL	41
Tabela 4	– Números buscados (NINGTYAS; MUTIARA, 2010)	63
Tabela 5	– Busca do número 1500	63
Tabela 6	– Tabela de Operadores com base na tabela de Ömer (2000)	77
Tabela 7	– Funções definidas em QCL com base nas tabelas de Ömer (2000)	79
Tabela 8	– Equivalência entre escalas	81

Sumário

1	INTRODUÇÃO	13
2	INTRODUÇÃO A MECÂNICA QUÂNTICA	17
2.1	ÁLGEBRA LINEAR	17
2.1.1	Álgebra Modular	18
2.1.2	Base e Independência Linear	19
2.1.3	Operadores Lineares e Matrizes	21
2.1.4	Autovetores e Autoestados	23
2.1.5	Notação de Dirac e base computacional	24
2.2	POSTULADOS DA MECÂNICA QUÂNTICA	25
2.2.1	Superposição de Estados	29
2.2.2	Paralelismo Quântico	29
2.2.3	Emaranhamento Quântico	30
3	INTRODUÇÃO À LINGUAGEM DE PROGRAMAÇÃO	33
3.1	CLASSIFICAÇÃO DAS LINGUAGENS DE PROGRAMAÇÃO	35
3.2	LINGUAGEM DE PROGRAMAÇÃO QUÂNTICA PARA COMPUTADORES CLÁSSICOS	36
3.3	LINGUAGEM DE PROGRAMAÇÃO QUÂNTICA QCL	37
4	COMPUTAÇÃO QUÂNTICA	45
4.1	MÁQUINA DE TURING	46
4.2	ESTADOS DE UM QUBIT	47
4.3	ALGORITMOS	52
4.3.1	Algoritmo de Grover	53
5	CONSIDERAÇÕES FINAIS	65
	Referências	67
	APÊNDICES	71
	APÊNDICE A – QCL	73
A.1	INSTALAÇÃO DO QCL	73

ANEXOS	75
ANEXO A – TABELAS DE OPERADORES QCL	77
ANEXO B – TABELA DE FUNÇÕES	79
ANEXO C – TABELA DE EQUIVALÊNCIA ENTRE DECIMAIS, BINÁRIO E HEXADECIMAL	81

1 INTRODUÇÃO

A busca para solução de problemas é o que move o desenvolvimento da ciência de uma forma geral. Frequentemente a ciência nos apresenta teorias e respostas à problemas que já haviam sendo pesquisados há algum tempo, entretanto essas novas descobertas podem levar a novas inquietações. Esse movimento de responder questionamentos “antigos”, e concomitantemente gerar novas indagações é o que torna a ciência dinâmica como a conhecemos (LIMA; MIOTO, 2007).

Tem-se como exemplo da ciência dinâmica o desenvolvimento tecnológico, evolui lado a lado com o desenvolvimento científico. O conhecimento científico está totalmente atrelado ao conhecimento tecnológico. O fato é, em um determinado momento um não consegue seguir adiante sem o outro, tornando-o um movimento cíclico entre ambos. Quando se avança cientificamente para produzir uma nova tecnologia, se constrói uma nova tecnologia (ferramenta) para se avançar cientificamente. Os cientistas de forma geral, costumam dar contribuições importantes aos avanços tecnológicos, tendo como exemplo disso a construção de aparatos (tecnológicos) para poder testar teorias (científicas) (MORAIS, 2010). As vezes esses aparatos ganham outras funções e acabam disseminados na sociedade: um exemplo que destaca-se é a invenção do computador, que inicialmente era uma ferramenta desenvolvida para realizar pesquisas científicas, e que acabou ganhando outras funcionalidades. Aquele que um dia foi desenvolvido com intuito de ser somente um aparato científico, hoje tornou-se uma importante ferramenta tecnológica.

A humanidade sempre buscou construir equipamentos ou ferramentas que auxiliassem na realização das tarefas diárias. Equipamentos que auxiliam o homem a realizar operações matemáticas, também fazem parte deste conjunto. Strathern (2000), cita inúmeros desses aparatos, desde o Ábaco a 4000 anos a. C., até a idealização de uma máquina computável, a máquina de Turing proposta por Alan Turing em um artigo de 1936. Neste intervalo de tempo o próprio Strathern (2000), cita inúmeras outras invenções ou aparatos que foram construídos para o intuito mencionado anteriormente. Algumas destas construções foram: a máquina inventada por William Schickard em 1623; a Régua de Cálculo, criada por William Oughtred; a matemática binária em 1623 criada pelo matemático Gottfried Leibniz; e ainda as Máquinas da Diferença N°1 e N°2 criadas por Charles Babbage em 1823. Lembrando ainda que foram feitas outras contribuições importantes feitas neste intervalo de tempo por outros matemáticos e cientistas.

Todos os equipamentos criados entre o período de 4000 anos a.C. até o final do século XVIII, tinham natureza analógica, pois seu funcionamento era mecânico, ou seja, dependia da movimentação de manivelas e rodas dentadas, entre outros aparatos, para

funcionarem. No ano de 1890 Herman Hollerith, criou a primeira máquina digital de que se tem conhecimento, que ficou conhecida como “máquina de censo”. A máquina de censo é considerada digital, pois apresenta funcionamento eletrônico, por esse motivo ela é considerada um “divisor de águas” tecnológico. O surgimento da primeira máquina eletrônica abriu um novo horizonte de estudos.

Observou-se anteriormente, fragmentos da história da construção da máquina de calcular que hoje denomina-se “Computador Clássico.” Este funciona com eletricidade, onde a sua unidade de armazenamento mais básica é o bit. Os valores associados a 1 e 0 da matemática binária são os estados de “passar corrente”(1), “não passar corrente”(0) elétrica. O eletromagnetismo faz parte da área de estudos da Física clássica, por isso o computador que funciona a partir do que foi mencionado, ganha o nome de computador clássico.

O computador clássico possui análogo quântico que é baseado na mecânica quântica. Esta por sua vez, surgiu após uma revisão nos conceitos da Física clássica, onde observou-se uma certa discrepância na descrição de determinados fenômenos. Essa revisão deu origem a um novo ramo de estudo da Física, o qual ganhou o nome de Física Quântica. Os estudos que se iniciaram ao término do Século XX, culminaram na sua “descoberta”, por volta da metade da década de 1920, estudos realizados pelo cientista Max Planck, que estudava a “radiação de corpo negro” (JR, 2000).

A partir da Mecânica Quântica, surgiram várias aplicações aos seus estudos, e a Computação Quântica faz parte desse escopo (NIELSEN; CHUANG, 2003; AL-DAOUD, 2007). Quando o primeiro computador clássico foi construído, sabia-se tudo que ele era capaz de fazer e quais as suas implicações. Esperava-se que o mesmo acontecesse com o Computador Quântico (STRATHERN, 2000).

Para alguns estudiosos da teoria da informação, ficou claro que era só uma questão de tempo para que a computação deixasse de obedecer a mecânica clássica, pois, com o passar do tempo se nota que a necessidade de espaço para os componentes de um computador vem caindo de forma rápida (OLIVEIRA, 2007). Nesse ponto pode-se citar a Lei de Moore ¹, que descreve o quanto um processador aumenta sua capacidade de processamento sem aumentar o seu tamanho, implicando que os componentes que formam o computador tornem-se cada vez mais reduzidos (MOORE; others, 1975).

Como foi citado, a computação quântica surgiu como uma área da mecânica quân-

¹ A Lei de Moore ganha o status de lei porque quando Gordon Moore fez a sua descrição ele constata um fato. Depois da sua descrição, as empresas podem ter usado a sua “lei” como uma meta. Cientificamente a Lei de Moore não tem validade nenhuma. O fato é, que mesmo se tratando de uma descrição do que se tinha até o momento, e subsequentemente uma tentativa de se prever o futuro, essa chamada lei continua a ser “obedecida”, chegando ao ponto em que se tem componentes tão pequenos no processamento de informação que os efeitos quânticos interferem no processamento (MOORE; others, 1975).

tica, ela tem como intuito, descrever como um computador que obedece as leis da mecânica quântica. Como veremos na seção 4.3, os computadores, seja quântico ou clássico, obedecem uma lógica de funcionamento, essa lógica é baseada em algoritmos. Quando a computação quântica surgiu, logo apareceram as primeiras tentativas de escrever um algoritmo que seguisse a lógica de funcionamento de um computador quântico. Hoje aceita-se que o primeiro computador quântico construído foi aquele feito pela empresa canadense D-Wave. Embora muito tenha-se especulado sobre a sua natureza quântica, pesquisas indicam que ele apresenta funcionamento quântico (SHIN et al., 2014).

A ideia de algoritmo é a mesma, independentemente de se tratar de Computação Quântica ou Clássica. O que diferencia os algoritmos quânticos é que eles, para computar as suas tarefas, fazem uso de estados quânticos. Estes por sua vez, estão sujeitos aos postulados da mecânica quântica, que estão descritos na seção 2.2. A primeira tentativa de se escrever um algoritmo quântico foi na década 80 com os pesquisadores Deutsch e Jozsa, que o fizeram para ser implementado em uma Máquina de Turing não-determinística, que será abordada na seção 4.1. Entretanto apenas em 1994 é que o primeiro algoritmo quântico significativo para a computação quântica foi publicado: o algoritmo de Shor. Em 96, outro algoritmo fundamental ao estudo da computação quântica foi publicado, o algoritmo de Grover. E esses dois algoritmos formam a base para a computação quântica (NIELSEN; CHUANG, 2003; OLIVEIRA, 2007; GRECA; MOREIRA, 2001).

A proposta deste trabalho é implementar o algoritmo de Grover, um algoritmo quântico de busca, utilizando a linguagem de programação quântica QCL (Quantum Computation Language). Buscou-se melhor compreensão dos fundamentos da mecânica quântica que possibilitam o processamento de informação em sistemas quânticos e que em alguns casos seriam mais eficientes do que em computadores clássicos. Com este fim, será estudado a manipulação dos estados quânticos pelos comandos disponíveis pela linguagem QCL.

2 INTRODUÇÃO A MECÂNICA QUÂNTICA

O estudo da computação quântica exige um pré-conhecimento de alguns conceitos físicos e matemáticos, de forma especial da mecânica quântica e da álgebra linear. Inicialmente será feito uma revisão de conceitos da álgebra linear, e em seguida veremos os postulados da mecânica quântica e conceitos afins.

2.1 ÁLGEBRA LINEAR

Segundo [Boldrini \(1986\)](#), álgebra linear é o estudo dos espaços vetoriais e das operações lineares neles realizados. A álgebra linear tem aplicação em muitas áreas, como: física; química; biologia; computação e muitas outras, não restringindo sua aplicabilidade somente a computação quântica, ou a mecânica quântica ([GONCALVES; others, 2013](#)).

Para [Boldrini \(1986\)](#), um dos conceitos mais utilizados, é o conceito de espaços vetoriais. O espaço vetorial que mais interessa é o \mathbb{C}^n , que é o espaço de todas as n-uplas de números complexos. Os elementos dos espaços vetoriais são denominados vetores, que geralmente são representados por (x_1, \dots, x_n) , que os vetores podem ser representados de forma matricial pelas suas componentes, nesse caso tem-se:

$$x_1 = \begin{bmatrix} a_1 \\ \vdots \\ a_n \end{bmatrix} \quad (2.1)$$

Dentro de um espaço vetorial existem algumas operações definidas, neste caso a soma de dois vetores que estão no espaço \mathbb{C} dá origem a um único vetor, essa operação fica definida dessa forma

$$\begin{bmatrix} a_1 \\ \vdots \\ a_n \end{bmatrix} + \begin{bmatrix} b_1 \\ \vdots \\ b_n \end{bmatrix} = \begin{bmatrix} a_1 + b_1 \\ \vdots \\ a_n + b_n \end{bmatrix}, \quad (2.2)$$

as somas dispostas ao lado direito da igualdade são somas ordinárias de números complexos. Dentro de um espaço vetorial \mathbb{C} também tem-se definida a multiplicação por escalar, a qual segue a seguinte definição

$$c \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix} \equiv \begin{bmatrix} cx_1 \\ \vdots \\ cx_n \end{bmatrix} \quad (2.3)$$

onde c é um escalar, ou seja, um número complexo e a multiplicação à direita da igualdade são produtos ordinários de números complexos (NIELSEN; CHUANG, 2003).

O produto interno em um espaço vetorial complexo também é definido. Segundo Malajovich (2007), “produto interno é uma abstração que permite introduzir noções de comprimento e ângulo em espaços vetoriais (p. 15)”. Segundo Nicholson (2015), não observa-se diferença na multiplicação por escalar e soma no espaço \mathbb{C} , em relação a espaço \mathbb{R} , mas existem diferenças quanto ao comprimento de um vetor, pois dado um vetor no espaço complexo $z = 1 + i$, o seu comprimento calculado da mesmo forma que no espaço real fica $|z| = \sqrt{1^2 + i^2} = \sqrt{1 + (-1)} = 0$. Entretanto pode-se entender o produto interno complexo como sendo o seguinte: dados dois vetores no espaço complexo $Z = \{z_1, z_2, \dots, z_n\}$ e $W = \{w_1, w_2, \dots, w_n\}$, o produto interno usual $\langle Z|W \rangle$ é denotado por

$$\langle Z|W \rangle = z_1 \bar{w}_1 + z_2 \bar{w}_2 + \dots + z_n \bar{w}_n, \quad (2.4)$$

onde \bar{w} representa o conjugado de w . Isso leva a uma noção mais satisfatória de comprimento de um vetor no espaço complexo

$$\langle Z|Z \rangle = z_1 \bar{z}_1 + z_2 \bar{z}_2 + \dots + z_n \bar{z}_n = |z_1|^2 + |z_2|^2 + \dots + |z_n|^2, \quad (2.5)$$

implicando em

$$||Z|| = \sqrt{\langle Z|Z \rangle}. \quad (2.6)$$

2.1.1 Álgebra Modular

Existe de modo geral, uma certa naturalização da álgebra usual, ao ponto de pensar-se que é a única forma de operar os números. De fato, as técnicas da aritmética usual são bem comuns e têm aplicação de seus conceitos de forma mais natural. Um outro tipo de aritmética que é bastante útil é a álgebra modular ou aritmética modular. Para essa álgebra, o que interessa são os restos das divisões.

Pode-se observar a álgebra modular em alguns objetos do cotidiano, como por exemplo o relógio. O tipo de álgebra que se usa para determinar as horas é modular. Quando o relógio marca 8:00 horas, e em seguida passam-se 5 horas, ao invés do relógio marcar 13:00 horas, ele irá marcar 1:00 hora (no relógio de ponteiros). Isso acontece por que o relógio está modulado em 12, ou seja, quando o intervalo de 12 horas acaba, o relógio começa da 1 hora novamente.

A aritmética modular é usada também em sistemas de identificação, com os números associados aos mais variados documentos que são emitidos a uma pessoa: RG, CPF, título de eleitor, passaporte, carteira de trabalho, etc (PICADO, 2001).

Segundo Gauss (2012), para um número positivo inteiro n , dois números inteiros a e b , são ditos congruentes com módulo n . Essa ideia segue a seguinte relação

$$a \equiv b \pmod{n}. \quad (2.7)$$

Quando o resultado da subtração de a por b é um múltiplo de n , ele é chamado de módulo da congruência. Se a e b , são dois números inteiros e a relação 2.7 é obedecida, a razão de a por n tem exatamente o mesmo resto que a razão de b por n . Observa-se isso no seguinte exemplo: $2 = 5 = 8 = 11 \pmod{3}$, onde a razão de qualquer um dos valores citados pelo número 3 sempre resulta em um resto 2.

Uma das formas da álgebra modular que ressalta-se nesse momento é a aritmética de módulo 2, essa aritmética é utilizada nos sistemas computacionais, facilmente implementada em computadores, pela matemática binária. A aritmética modular é importante a sistemas computacionais, pois não existe a propagação dos números nas casas binárias da operação. A soma em binário tradicional, de dois valores como $01+01=10$, apresentam uma propagação no valor de 1 pelas casas do binário. Já na soma de módulo dois $10+10=00$, não há a propagação de um valor pelas casas binárias, como na soma binária comum.

Sem entrar em muitos detalhes, a soma de módulo 2 tem os seguintes valores:

$$\begin{cases} 0 + 0 = 0 \pmod{2} \\ 1 + 0 = 1 \pmod{2} \\ 0 + 1 = 1 \pmod{2} \\ 1 + 1 = 0 \pmod{2} \end{cases},$$

e são análogos a subtração

$$\begin{cases} 0 - 0 = 0 \pmod{2} \\ 1 - 0 = 1 \pmod{2} \\ 0 - 1 = 1 \pmod{2} \\ 1 - 1 = 0 \pmod{2} \end{cases}$$

(GAUSS, 2012; NEVES, 2001; NIELSEN; CHUANG, 2003; GONCALVES; others, 2013).

2.1.2 Base e Independência Linear

Para definir uma base linear, primeiramente deve-se definir independência linear. Definido o produto interno de u por v , se o produto interno de u por v for zero, isso quer dizer que os dois vetores são linearmente dependentes, pois $u = \alpha v$. Onde α é um escalar qualquer, o que implica que os vetores associados são simplesmente um múltiplo do outro. O produto escalar entre dois vetores, relaciona os vetores associados ao ângulo que o mesmos fazem entre si. Quando o produto interno de dois vetores é 0, significa que eles são paralelos ou até mesmo que são vetores coincidentes. Se o produto interno é 0, os vetores associados não são considerados base de um espaço.

A concepção de base de um espaço vetorial consiste em escolher um conjunto de vetores de base que seja o menor possível, ou seja, um conjunto de elementos suficientes e necessários na geração do espaço em questão. Exemplos que são bastante comuns que pode-se citar é o plano cartesiano e o espaço cartesiano, onde podemos representar objetos de duas e três dimensões respectivamente. Esses espaços podem ser gerados por dois ou três vetores, se retirarmos qualquer um dos vetores de ambos os espaços, não podemos mais gerar o espaço todo. Uma propriedade importante dos vetores da base, é que eles devem ser linearmente independentes, como mencionado.

Afirma-se que determinada uma base, essa base gera um espaço vetorial, de acordo com as propriedades dos vetores geradores, que são a base.

Basicamente, a definição de espaços vetoriais, para por exemplo os reais, segundo [Steinbruch \(1995\)](#), é: “[...] será constituído um conjunto de de nodas as n-uplas ordenadas e representado por R^n , isto é (p. 15):

$$R^n = \{(x_1, x_2, \dots, x_n); x_i \in R\}'' \quad (2.8)$$

Os espaços vetoriais permitem apenas algumas operações. Citamos algumas das operações permitidas no espaço complexo na seção anterior, a completude deste espaço depende das operações que este espaço permite. Basicamente, todas as operações que estão definidas dentro espaço real também estão definidas no espaço complexo. A não ser, como citado, pela definição de comprimento, ou módulo de um vetor, isso implica na definição um produto interno distinto. Neste contexto, as operações definidas dentro dos espaços complexos deve obedecer as propriedades de soma e multiplicação, dentre outras operações, da matemática binária.

A definição geral de espaço vetorial é análoga a equação 2.8, com a ressalva de que, ao invés de ter a definição para os reais, tem-se a definição para qualquer espaço. Dessa forma:

$$E^n = \{(e_1, e_2, \dots, e_n); e_i \in E\}'', \quad (2.9)$$

com E , sendo um espaço vetorial qualquer.

Pode-se determinar os vetores que formam o espaço, ou seja, que são os vetores de base do espaço em questão. Se as condições citadas forem respeitadas, qualquer outro vetor que pertença a esse espaço pode ser escrito como combinação linear dos vetores da base ([NIELSEN; CHUANG, 2003](#)). Para melhor observar atentemo-nos a um exemplo: pode-se escolher dois vetores que gerem o espaço R^2 , esses dois vetores podem ser os vetores u_1 e u_2 que tem o valor igual a

$$u_1 \equiv (1, 0); \quad u_2 \equiv (0, 1), \quad (2.10)$$

que como vê-se também podem ser escritos na forma

$$u_1 = \begin{bmatrix} 1 \\ 0 \end{bmatrix}; \quad u_2 = \begin{bmatrix} 0 \\ 1 \end{bmatrix}. \quad (2.11)$$

Um segundo exemplo que pode-se citar é o espaço R^n que poderia ser escrito da forma

$$u_1 = \begin{bmatrix} 1 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \quad u_2 = \begin{bmatrix} 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \quad u_3 = \begin{bmatrix} 0 \\ 0 \\ 1 \\ \vdots \\ 0 \end{bmatrix} \quad u_n = \begin{bmatrix} 0 \\ 0 \\ 0 \\ \vdots \\ 1 \end{bmatrix}. \quad (2.12)$$

Ambas as bases citadas são ditas canônicas, devido a sua naturalidade. Os espaços mencionados tem inúmeras possibilidades de bases que os geram, resume-se isso a todos os vetores que satisfazem a relação de ortogonalidade:

$$\langle u, v \rangle = 0. \quad (2.13)$$

A teoria da Computação Quântica faz uso de um espaço que é denominado espaço de Hilbert, que será explicado na seção 2.2. Observa-se que nos dois exemplos citados acima, os vetores contemplavam duas características que são fundamentais: a primeira é que em ambos os exemplos, os vetores definidos como base de cada espaço geravam todo o espaço mencionado, como havia-se dito; a segunda, é que os vetores são linearmente independentes, isso quer dizer que, não consegue-se escrever qualquer um dos vetores da base de qualquer um dos espaços a partir dos outros, ou seja, eles não obedecem as relações

$$u_1 = c \cdot u_2 \quad (2.14)$$

$$u_1 = c \cdot u_2 + d \cdot u_3, \quad (2.15)$$

ou qualquer outra variação destas, onde os valores de c e d podem assumir qualquer valor dentro dos reais.

2.1.3 Operadores Lineares e Matrizes

Para entender o conceito de operador linear primeiramente existe a necessidade de definir-se o que é função. Segundo [Anton e Busby \(2006\)](#) as funções podem ser definidas como:

Dado um conjunto D de entradas permitidas, dizemos que uma função f é uma regra que associa uma única saída a cada entrada de D ; o conjunto D é denominado domínio de f . Denotando a entrada por x , escrevemos $f(x)$ [...] para a saída correspondente. A saída também é denominada o valor de f em x ou a imagem de x por f , e dizemos que f aplica ou leva

x em $f(x)$. É costume denotar a saída por uma única letra y e escrever $y = f(x)$. O conjunto de todas as saídas y que resultam quando x varia sobre o domínio é denominado a imagem de f (p. 269).

Uma função onde os valores de entrada e de saída são vetores, é dita transformação. As transformações podem ser realizadas através de operadores, que podem ser representados na forma matricial. Transformações feitas a partir de matrizes são um caso especial das transformações unitárias, e a ação de um operador matricial pode ser denotada da seguinte maneira (ANTON; BUSBY, 2006):

$$T : R^n \longrightarrow R^m. \quad (2.16)$$

Onde a transformação T leva o vetor que está no R^n para o seu correspondente no espaço R^m . Um operador muito importante da álgebra linear é denominado operador identidade (I), que tem a ação de levar um vetor v nele mesmo, ou seja, $I(v) = v$. Na forma matricial e no espaço bidimensional, ou seja, o R^2 o operador identidade fica:

$$I = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}. \quad (2.17)$$

O operador identidade está definido em todos os espaços vetoriais conhecidos, e sua ação sempre ocorre da mesma maneira.

Existem duas classes de operadores lineares que se destacam, a primeira é a classe dos operadores adjuntos e a segunda são os operadores Hermitianos.

Segundo Piza (2003), a definição de operadores adjuntos permite a definição de dois tipos de distintos de operadores unitários. A primeira é classe dos operadores unitários, esse tipo de operador é identificado pela ação $u^\dagger = u^{-1}$, ou ainda

$$uu^\dagger = u^\dagger u = I, \quad (2.18)$$

onde I é o operador identidade. Operadores unitários preservam o produto interno dos vetores, consequentemente preserva a norma. Um outro fato importante, é que o produto de operadores lineares resulta em outro operador linear. O produto de um operador linear por uma base ortonormal, leva em outra base, também ortonormal, por exemplo:

$$Ia = I \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \quad ; \quad Ib = I \begin{pmatrix} 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \end{pmatrix} \quad (2.19)$$

A segunda classe de operadores adjuntos são os operadores de projeção, que são identificados pelas propriedades:

$$P^\dagger = P \quad \text{e} \quad P^2 = PP = P. \quad (2.20)$$

Utilizando-se desse tipo de operador, um vetor pode ser escrito como combinação linear de dois outros vetores ortogonais, ou seja, pode ser decomposto em coordenadas.

2.1.4 Autovetores e Autoestados

Dada uma transformação linear de um espaço vetorial para ele mesmo, existe um espaço vetorial A , definido então uma transformação $T : A \longrightarrow A$, essa transformação gera um questionamento: quais vetores pertencentes a A seriam levados nele mesmo? (STEINBRUCH, 1995)

Frente a isso utiliza-se o recurso de um exemplo. Então tem-se uma transformação:

$$T : R^2 \longrightarrow R^2 \quad (2.21)$$

onde a lei da transformação é dada por $T(x, y) = (x, y)$. Nesse caso pode-se notar que todos os vetores serão levados neles mesmos. Em um segundo exemplo pode-se notar uma variação.

$$T : R^2 \longrightarrow R^2 \quad (2.22)$$

Com a lei de transformação sendo $T(x, y) = (x, -y)$, pode-se notar que a componente x não será alterada, ao contrário a componente y terá o seu sinal invertido. Essa relação também pode ser escrita na forma matricial, ficando:

$$\begin{bmatrix} x \\ y \end{bmatrix} \longrightarrow \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \end{bmatrix} \quad (2.23)$$

é possível notar que todos os vetores que estão sobre o eixo x , no plano cartesiano, não serão alterados, pois tem componente $y = 0$.

$$\begin{bmatrix} x \\ 0 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} = \begin{bmatrix} x \\ 0 \end{bmatrix} \quad (2.24)$$

Ao fazer-se uma análise mais detalhada, chega-se a conclusão que a única solução para essa transformação é $(x, 0)$. Utilizando a equação 2.23, resulta-se no seguinte sistema:

$$\begin{cases} x + 0y = x \\ 0x - y = y \end{cases}$$

Sendo a única solução possível para esse sistema é $(x, 0)$.

Um segundo ponto a ser analisado, ocorre quando tem-se uma transformação de $T : A \longrightarrow A$, e quer-se saber quais dos vetores são levados em múltiplos deles mesmos. De forma mais sintetizada procura-se

$$T(a) = \lambda(a), \quad (2.25)$$

isto é, quando o valor de λ é 1, é o vetor que assume o mesmo valor de antes da transformação, onde λ é o autovalor do estado. Caso λ não assuma o valor igual 0 ou 1, tem-se um vetor na mesma direção, que pode diferir tanto em módulo quanto em sentido, o que caracteriza uma multiplicidade de vetores.

Para tomar-se conhecimento dos nomes e conceitos relacionados as transformações, define-se λ , que será denominado autovalor de a na equação 2.25, bem como, a é o autovetor associado a λ (BOLDRINI, 1986).

Para calculá-los, tanto autovetor como autovalor, pode-se utilizar da equação característica, que é definida como

$$T(\lambda) = \det|U - \lambda I|, \quad (2.26)$$

em que \det é o conceito determinante de matrizes, e U é um operador que relaciona o autovalor com autovetor. As soluções da equação característica para $T(\lambda) = 0$ são os autovalores do operador U . Segundo o teorema fundamental da álgebra, todo polinômio tem ao menos uma solução, ou seja, existe pelo menos um autovetor e um autovalor associado correspondente a U . Temos ainda o autoespaço, que nada mais é onde o conjunto de vetores correspondentes ao autovalor λ , e é nesse autoespaço que U atua.

Dada a matriz A :

$$A = \begin{bmatrix} 2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 2 \end{bmatrix} \quad (2.27)$$

a matriz A possui três autovalores iguais a 2, neste caso o espaço é dito degenerado (STEINBRUCH, 1995).

2.1.5 Notação de Dirac e base computacional

Segundo Bassalo (2006), o físico inglês Paul Adrien Maurice Dirac em 1930 descreveu uma matemática para ser utilizada no espaço de Hilbert. Essa notação define

$$|\psi\rangle \equiv \text{vetor ket} \quad (2.28)$$

e

$$\langle\psi| \equiv \text{vetor bra}. \quad (2.29)$$

O vetorket pode ser definido como:

$$|\psi\rangle = \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_n \end{bmatrix}, \quad (2.30)$$

onde a_n são as componentes do vetor. Já o vetor bra é definido como:

$$\langle\psi|^* = \langle\psi| = [a_1 \ a_2 \ \cdots \ a_n]. \quad (2.31)$$

As propriedades abaixo definem as operações permitidas a vetores em um determinado espaço vetorial, que foram definidas em seções anteriores, mas agora elas estão expressas segunda a notação de Dirac:

1. $(|\psi\rangle)^* = \langle\psi|$
2. $\langle\psi|\phi\rangle^* = \langle\phi|\psi\rangle$
3. $\langle\psi|(|\phi\rangle + |X\rangle) = \langle\psi|\phi\rangle + \langle\psi|X\rangle$
4. $\langle\psi|c\phi\rangle = c\langle\psi|\phi\rangle$
5. $\langle c\psi|\phi\rangle = c^* \langle\psi|\phi\rangle$

Conforme foi ilustrado anteriormente, vetores podem ser escritos da forma matricial, como na representação da notação de Dirac feita anteriormente. A base computacional quântica é definida em um espaço vetorial composto por dois vetores ortonormais. Esses vetores são $(1,0)$ e $(0,1)$. Pode-se dizer que $|\phi\rangle = (1,0)$ e $|\theta\rangle = (0,1)$, e essa é a base ortonormal da computação quântica, por isso, geralmente, os vetores $|\phi\rangle$ e $|\theta\rangle$, ganham uma nomenclatura diferenciada. Que é:

$$|\phi\rangle = |0\rangle = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \quad ; \quad |\theta\rangle = |1\rangle = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \quad (2.32)$$

o espaço possui duas dimensões.

Segundo Oliveira (2007), existem vários espaços possíveis a se trabalhar dentro da teoria quântica, mas o que nos desperta maior interesse é o espaço q-bit. Esse espaço tem n dimensões, o qual podemos definir a sua base ortonormal como sendo $|0\rangle$ e $|1\rangle$, e a partir desses vetores podemos escrever qualquer outro, simplesmente fazendo

$$|\psi\rangle = a|0\rangle + b|1\rangle, \quad (2.33)$$

em que a e b são números complexos. Podemos observar a representação gráfica de $|\psi\rangle$ na representação de Bloch na figura 4. Para esse espaço, existe a condição de que $|\psi\rangle$ seja unitário, o que implica em $\langle\psi|\psi\rangle = 1$, da mesma forma $|a|^2 + |b|^2 = 1$. Essas são as condições de normalização.

2.2 POSTULADOS DA MECÂNICA QUÂNTICA

O debate sobre Mecânica Quântica inicia-se com os seus postulados. Estes por sua vez, norteiam os estudos teóricos e experimentais na mecânica quântica. Como a Computação Quântica se baseia na mecânica quântica, então as teorias sobre a computação quântica obedecem seus postulados.

O **primeiro** postulado se refere ao espaço onde se trabalham os principais conceitos relacionados a mecânica quântica, ou seja, relaciona um sistema físico qualquer a um espaço vetorial complexo, que é o espaço de Hilbert (NIELSEN; CHUANG, 2003).

Segundo Griffiths (2007), o espaço de Hilbert é um espaço não-trivial, o que faz ser essencial definir-se bem as operações permitidas dentro deste. Relembrando a relação de algumas seções acima (2.1), onde tem-se a descrição de um vetor na forma matricial, pode-se reescrever a referida equação, pois também pode-se representar um vetor da seguinte forma:

$$|\alpha\rangle \longrightarrow a = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}, \quad (2.34)$$

como já mencionado, operadores lineares atuam nos vetores na forma de transformações lineares.

O produto interno entre dois vetores $\langle \alpha | \beta \rangle$, gera como resultado um número complexo, quando esses vetores estão contidos no espaço de Hilbert. Nesse caso

$$\langle \alpha | \beta \rangle = a_1^* b_1 + a_2^* b_2 + \cdots + a_n^* b_n \quad (2.35)$$

com isso a sua norma fica denotada como $\|v\| = \sqrt{\langle \alpha, \beta \rangle}$.

As transformações lineares neste espaço são descritas na forma de matrizes, quando aplicada sobre um vetor pertencente ao espaço produzirá um novo vetor, obedecendo as regras da transformação, pode-se representar como

$$|\beta\rangle = T|\alpha\rangle \longrightarrow \begin{bmatrix} t_{11} & t_{12} & \cdots & t_{1n} \\ t_{21} & t_{22} & \cdots & t_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ t_{n1} & t_{n2} & \cdots & t_{nn} \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_n \end{bmatrix}, \quad (2.36)$$

Para facilitar o trabalho, é necessário por limites nesses espaços. Para fazer isso, escolhe-se um subespaço que pode ser por exemplo o conjunto de todas as funções de x , ainda sim ele é muito amplo. Por se tratar de um espaço complexo, o espaço de Hilbert permite dentro dele a definição de funções de onda.

O produto interno entre duas funções $f(x)$ e $g(x)$ é definido como

$$\langle f | g \rangle \equiv \int_a^b f(x)^* g(x) dx \quad (2.37)$$

se ambas as funções pertencerem ao espaço de Hilbert, então a solução a esse produto interno existe.

O produto interno de $f(x)$ por ele mesmo é definido como:

$$\langle f | f \rangle = \int_a^b |f(x)|^2 dx. \quad (2.38)$$

Uma certa comutatividade também pode ser observada quando falamos de espaços de Hilbert. Nesse caso:

$$\langle g|f \rangle = \langle f|g \rangle^* . \quad (2.39)$$

Uma função está normalizada quando o produto interno dela com ela mesma é igual a 1 (GRIFFITHS, 2007).

Sabe-se que qualquer sistema de numeração que trabalha-se, tem um espaço vetorial definido. O postulado 1 da mecânica quântica se refere exatamente a isso, só que ao invés de termos um conjunto qualquer de números, temos um sistema físico isolado. A esse sistema pode ser associado um espaço vetorial complexo com produto interno definido, esse estado pode ser chamado também de espaço de estados do sistema. Esse sistema, pode ser descrito por um vetor, pertencente a ele, e que pode ser chamado de vetor de estado, que nada mais é que um vetor unitário no espaço de estados (NIELSEN; CHUANG, 2003).

No **postulado 2**, tem-se a descrição de como o sistema quântico fechado evolui, que é na forma de uma transformação unitária. O estado $|\psi \rangle$ de um determinado sistema em um tempo t_1 , está diretamente ligado ao estado $|\psi' \rangle$ em um tempo t_2 . Logicamente isso em um mesmo sistema, onde a relação entre estes dois estados é dado por uma operador unitário U (NIELSEN; CHUANG, 2003).

O postulado 2, funciona para descrever sistemas isolados, ou seja, sistemas que não tenham nenhum tipo de contato com sistemas externos. De fato, isso é muito difícil de acontecer, pois sistemas sempre interagem uns com os outros. Se utilizando de boas aproximações como ferramenta é possível descrever alguns sistemas como fechados, e obter ótimos resultados. Outra perspectiva pode ser levada em consideração, a priori, todos os sistemas, em algum nível, podem ser descritos como isolados, e depois entram em contato com um sistema maior, o universo, que é isolado e não interage com ninguém, então ele pode ser descrito na forma de transformações unitárias.

Podemos descrever um sistema quântico fechado pela equação de Schrödinger

$$i\hbar \frac{d|\psi \rangle}{dt} = K|\psi \rangle \quad (2.40)$$

onde \hbar é uma constante, que está associada a constante de Plank (h), o seu valor pode ser determinado experimentalmente (NIELSEN; CHUANG, 2003).

$$|\psi' \rangle = U|\psi \rangle . \quad (2.41)$$

A partir da equação de Schrödinger pode-se obter o operador U do enunciado anterior, ou seja, não é qualquer operador, e sim um operador que é obtido da resolução da equação de Schrödinger. Um exemplo de operador unitário que pode vir a ocupar o lugar de U na equação (2.41) é o operador Hadamard, embora não forneça uma evolu-

ção temporal sobre o estado observado, ele é de grande valia ao estudo da Computação Quântica.

No estudo da física clássica, pode-se determinar a posição de determinada partícula para cada instante de tempo em que ela descreve um determinado tipo de movimento, incluindo o instante do tempo “agora”. Na mecânica quântica, determinar a posição da partícula não é o problema em si, se for tomada uma região considerável do espaço. Devido ao fato de partículas atômicas e subatômicas terem uma dinâmica de evolução espacial um tanto quanto conturbada, ao ponto de algum milésimos de segundo após realizar-se uma medida sobre ela, não se faz a mínima ideia de onde a partícula se encontra. O que pode ser feito, é o cálculo da amplitude de probabilidade de onde a partícula tem mais chances de ser medida.

O **postulado 3** mostra como as informações interferem no estado de um sistema quântico. A leitura externa interfere no resultado final. Pode-se citar o exemplo de um carga elétrica pontual, onde pretende-se determinar qual o seu estado. Para fazer a medida desta carga pontual, todas as maneiras de se fazer essa medida, que são conhecidas, envolvem a adição de uma carga de prova no sistema. A adição dessa carga de prova acaba por interferir no estado inicial do sistema, fazendo com que não se conheça o estado inicial do sistema experimentalmente. Diferentemente de sistemas físicos clássicos onde não existe nenhuma interferência na hora em que o resultado é lido. Totalmente diferente da mecânica quântica quântica, o estado sofre uma interferência da própria medida não sendo mais o mesmo.

Segundo [Conceitos... \(2003\)](#), existe a perda de informação quando um sistema físico quântico é medido. Deve-se isso, em partes, pela dualidade onda partícula. Efeitos ondulatórios das partículas, quando medidos, se perdem.

Para estabelecer as condições das medidas, é possível estabelecer um conjunto de operadores M_m , onde o valor m refere-se ao valor de leitura possível, que ganha o nome de observável. O valor de m é definido pela base que é utilizada para codificar o sistema. Na base computacional a base é dada por $|\psi\rangle = a|0\rangle + b|1\rangle$. Automaticamente os observáveis são definidos como $m = \{0, 1\}$.

Conforme [Oliveira \(2007\)](#), a medição é um experimento probabilístico que retorna m como uma determinada probabilidade, que pode ser definida como

$$p(m) = \langle \Psi | M_m^\dagger M_m | \Psi \rangle, \quad (2.42)$$

onde os operadores de medição da base computacional são $M_0 = |0\rangle\langle 0|$ e $M_1 = |1\rangle\langle 1|$. Ainda faz parte deste postulado o estado após a medição, que é definido como sendo

$$|\Psi'\rangle = \frac{M_m |\Psi\rangle}{\sqrt{P(m)}} \quad (2.43)$$

2.2.1 Superposição de Estados

A superposição de estados é um dos conceitos mais conhecidos da Mecânica Quântica, e tem papel fundamental na Computação Quântica, afirmando que um sistema pode assumir vários valores ao mesmo tempo. Um exemplo físico a isso é as duas polarizações distintas de um fóton, que quando não é medido pode assumir qualquer valor.

A grosso modo, dado um sistema físico quântico, como por exemplo dois elétrons orbitando em torno de núcleo atômico. Esses dois elétrons não podem ocupar o mesmo estado quântico, por definição. Eles se encontram em um mesmo orbital do átomo, pois devem ocupar o estado de menor energia. A única propriedade física que os diferencia, ou que os coloca em estados quânticos distintas são os seus respectivos estados spins, um pode apresentar spin $-\frac{1}{2}$, enquanto o outro apresenta spin $+\frac{1}{2}$. Se não for realizada nenhuma medida sobre este sistema quântico, qual é o elétron que apresenta spin $-\frac{1}{2}$? E qual apresenta spin $+\frac{1}{2}$? Fisicamente é impossível dizer. Além disso, esses dois elétrons podem estar continuamente invertendo os seus spins, ou seja, uma troca simultânea de estados. Com isso, pode-se observar que um sistema quântico como esse pode assumir uma quantidade enorme de estados, além disso, os estados dos elétrons do exemplo, estão superpostos. Um interfere no estado do outro. Voltando ao exemplo anterior, dos spins, objetiva-se medir apenas um dos elétrons, cada um deles tem 50% de probabilidade de ser medido.

Esse conceito está relacionado ao Postulado 1, onde descreve-se o estado $|\psi\rangle$. Um importante operador que também está relacionado com esse conceito é o operador Hadamard (equação ??), é ele que produz a superposição de estados no computador quântico, o operador Hadamard assume a seguinte formulação na notação de Dirac

$$H = \frac{1}{\sqrt{2}}(|0\rangle\langle 0| + |0\rangle\langle 1| + |1\rangle\langle 0| - |1\rangle\langle 1|). \quad (2.44)$$

O operador Hadamard, gera um estado de particular interesse a computação quântica. Uma vez que a superposição de estados é algo de interesse. Esse operador é utilizado para produzir estes estados, como mencionado, é vastamente utilizado, e faremos uso dele neste estudo ([OLIVEIRA, 2007](#)).

2.2.2 Paralelismo Quântico

Segundo [Alves \(2015a\)](#), o paralelismo quântico pode ser entendido como:

É a principal vantagem dos computadores quânticos em relação aos computadores clássicos. O paralelismo quântico permite aos computadores quânticos avaliarem uma função $f(x)$ para muitos valores diferentes de x simultaneamente.

[...] O funcionamento dos componentes dos computadores atuais é baseado nas propriedades quânticas da matéria, contudo, os bits, unidades

fundamentais de processamento, são clássicos, dado que podem estar apenas no estado $|0\rangle$ ou no estado $|1\rangle$. Em contraposição, os bits de um computador quântico, ou qubits, poderiam ser colocados em estados que são superposições coerentes do estado $|0\rangle$ e do estado $|1\rangle$ (p. 15-16).

O paralelismo é um outro efeito quântico muito importante para a computação quântica. Basicamente o paralelismo é a capacidade de um sistema quântico realizar várias funções ao mesmo tempo. Um computador quântico pode realizar ao mesmo tempo o cálculo de uma função $f(x)$ para muitos valores de x . Um exemplo de algoritmo que se apropria claramente do paralelismo quântico é o Algoritmo de Deutsch. É fato que os algoritmos em geral usam o paralelismo, e é ele que permite fazer questionamento quanto a potencialidade da computação quântica de forma geral (SILVA, 2003).

2.2.3 Emaranhamento Quântico

Segundo Bulnes (2005), quando dois ou mais estados quânticos interagem, o estado final de um vai depender dos outros estados. A ideia de emaranhamento parte desta situação.

Exemplos de estados emaranhados podem ser descritos pelos estados de Bell:

$$|\psi^\pm\rangle = \frac{1}{\sqrt{2}}(|00\rangle \pm |11\rangle) \quad (2.45)$$

$$|\phi^\pm\rangle = \frac{1}{\sqrt{2}}(|01\rangle \pm |10\rangle) \quad (2.46)$$

que são vetores no espaço de Hilbert, e correspondem a sistemas quânticos formados por duas partículas.

Para Oliveira e Oliveira (2008), os estados de Bell são exemplos de Emaranhamento Quântico, mas somente com o surgimento da teoria da informação é que esse conceito começou a ser melhor estudado. O emaranhamento tem várias aplicações, tais como: criptografia quântica, teletransporte de estados e a aceleração exponencial de algumas computações, entre outras.

Uma das manifestações do emaranhamento mais conhecidas, e também a mais citada nos mais distintos referenciais, é a criação de um par de fótons gêmeos, através de um processo atômico. Devido a conservação do momento linear, os fótons devem ser emitidos em direções contrárias, e pela conservação do momento angular devem ter as suas polarizações ortogonais. Decide-se passar um destes fótons por um polarizador, então este assume a devida polarização, no mesmo instante o fóton gêmeo assume uma polarização ortogonal, devido uma “ação a distância fantasmagórica” que foi uma das afirmações feitas por Einstein.

Pode-se chegar aos estados de Bell aplicando os operadores Hadamard e XOR . O operador XOR , faz parte das portas Not controladas quânticas, a sua ação computacional é negar os estados dos q-bits de acordo com uma condição. Essa condição depende não apenas do estado de um q-bit, mas sim de todos os q-bits que estão envolvidos na operação. Segue uma tabela da ação computacional de XOR (1).

Tabela 1 – Ação computacional de XOR

x_1	x_2	$x_1 \text{ XOR } x_2$
0	0	0
1	0	1
0	1	1
1	1	0

A aplicação dos operadores Hadamard e XOR seguem esta mesma ordem no estado de dois q-bits ($|00\rangle$), da seguinte maneira (BULNES, 2005; ALVES, 2015b):

$$H|00\rangle = \frac{|00\rangle + |10\rangle}{\sqrt{2}} \quad (2.47)$$

e

$$XOR(H|00\rangle) = \frac{|00\rangle + |11\rangle}{\sqrt{2}} = |\psi^+\rangle, \quad (2.48)$$

esse estado representa, como já mencionado, os estados emaranhados de duas partículas, mas também representa um importante estado para a computação quântica. Com isso, observa-se que na equação 2.47 não ocorre o emaranhamento, mas na equação 2.48, o emaranhamento se faz presente.

3 INTRODUÇÃO À LINGUAGEM DE PROGRAMAÇÃO

Linguagem de programação, é a forma com que os programadores se comunicam com o computador ou componente eletrônico semelhante. Existe na computação, uma infinidade dessas linguagens e basicamente todas elas tem um objetivo comum, que é uma forma de escrever algoritmos. Pode-se perceber a importância das linguagens de programação, basicamente tudo que conhecemos que tenha um funcionamento eletrônico e desempenha alguma tarefa, seja mecânica ou simplesmente de caráter visual como por exemplo o que o computador mostra em seu monitor. Dependem e dependem em alguma etapa de sua construção ou funcionamento de algum tipo de programação.

Na seção 4.3 descreve-se o que é um algoritmo, e como é basicamente o seu funcionamento. Ao discorrer da abordagem, algumas dúvidas podem ter surgido. Uma delas pode ter sido, **“como escrever um algoritmo em um computador clássico?”**. Essa transcrição, dependendo do nível da abordagem, não é tão complexa, mesmo assim não pode-se perder a ideia de que um computador nada mais é do que um sistema físico programável, ou passível de execução de algum algoritmo (JOSE; PIQUEIRA; LOPES, 2013).

Existe uma necessidade que os componentes do computador se comuniquem uns com os outros. Por exemplo, quando os dados inseridos no teclado são armazenados na memória; quando o processador precisar realizar operações aritméticas ele “busca” dados na memória. Para que se estabeleça uma comunicação entre os componentes é necessário que seja estabelecida uma linguagem, e por definição essa linguagem precisa apresentar símbolos básicos (EVARISTO, 2000).

A linguagem que os componentes do computador utilizam para se comunicarem é a linguagem de máquina, também conhecida como linguagem binária. Como a comunicação entre os entes que formam o computador devem ser fenômenos físicos, os cientistas que conceberam os computadores atuais estabeleceram dois símbolos básicos. Feito desta forma pela facilidade de definição de dois estados distintos e inconfundíveis na física, como: passar corrente elétrica/não passar corrente elétrica, estar magnetizado/não estar magnetizado. Podendo ser definido qualquer um desses estados físicos como sendo um desses símbolos, assim a linguagem de máquina tem apenas dois símbolos. Cada símbolo é denominado bit (binary digit) e universalmente representados por 0 e 1 (EVARISTO, 2000).

Quando a computação surgiu, havia a necessidade dos programadores programa-

rem utilizando a linguagem de máquina, dava-se a necessidade que se conhecesse a sequência de “0” e “1” que fizesse o computador executar a tarefa desejada. Para programas escritos dessa forma dava-se o nome de programas objeto. Na atualidade temos uma série de programas escritos em uma linguagem mais próxima a utilizada por nós, pessoas, para se comunicar uns com os outros. Esse tipo de linguagem é chamada de linguagem de alto nível, onde o alto nível se refere que a linguagem se aproxima da linguagem utilizada pelo ser humano. Quanto mais uma linguagem se aproxima da linguagem de máquina, esta é dita linguagem de baixo nível. Um programa escrito em linguagem de alto nível é dito programa fonte ou simplesmente programa (EVARISTO, 2000).

Para que funcione um programa fonte deve ser traduzido em linguagem de máquina, e para isso existem dois tipos de programas, os interpretadores que traduzem os programas em partes, mais precisamente comando por comando, e os compiladores que traduzem o programa por completo, para a linguagem de máquina. Os compiladores recebem um programa fonte e fornece um programa objeto, a compilação deste segundo gera então um programa que pode ser executado. Já os interpretadores, traduzem os comandos um a um para linguagem de máquina os executando em seguida, portanto os interpretadores não geram um programa objeto (EVARISTO, 2000).

Toda a computação está voltada para um ramo específico, que de fato é fundamental para a mesma. A programação de um computador, é simplesmente a causa que torna essa ferramenta tão extraordinária. Além de tudo isso, a programação é a base para se testar as potencialidades do hardware, e aí pode-se levantar o questionamento: como testar os limites da computação quântica? Da mesma forma, deve-se programá-la para verificar-se a potencialidade do hardware. Isso leva a uma nova indagação: como programar computadores quânticos sem ter acesso a um? Essa é uma pergunta que pretende-se responder neste capítulo (JOSE; PIQUEIRA; LOPES, 2013).

Um programador, não precisa conhecer os processos físicos que o computador realiza para a execução do programa, mas ele precisa conhecer a lógica de funcionamento do computador. Analogamente, isso ocorre na computação quântica, o programador não precisa conhecer a física que está acontecendo, e sim somente compreender uma outra lógica de funcionamento. Essa nova lógica tem três passos (ÖMER, 2000):

- preparação dos estados iniciais;
- realização das transformações unitárias;
- execução das medidas;

3.1 CLASSIFICAÇÃO DAS LINGUAGENS DE PROGRAMAÇÃO

Para [Jose, Piqueira e Lopes \(2013\)](#), existem 2 tipos distintos de linguagens de programação. Cada uma delas tem uma lógica de funcionamento. Ao todo existem mais de 130 linguagens de programação, e cada um dos 2 tipos de linguagens estão explicados abaixo.

As linguagens de programação ditas linguagens imperativas ou procedurais, são construídas com declarações que mudam o estado global do programa ou as variáveis associadas a ele. Exemplos de linguagens imperativas são: Pascal, FORTRAN, C, C++ e Java. Atualmente são as mais utilizadas na programação quântica ([JOSE; PIQUEIRA; LOPES, 2013](#)).

Linguagens de programação ditas funcionais são o segundo grupo, e segundo [Jose, Piqueira e Lopes \(2013\)](#),

Linguagens funcionais utilizam um tipo de paradigma para programação de computadores em que o foco das funções é puramente nas entradas e saídas, tal qual ocorre na Matemática, em oposição às linguagens imperativas que alteram estados e dados. APL e Lisp são exemplos de linguagens funcionais clássicas.

Maymin apresentou, em 1996, um cálculo lambda com um foco probabilístico e outro quântico. Em 1997, ele destaca que o cálculo lambda quântico pode simular de maneira eficiente um autômato celular quântico, particionado em uma dimensão (p. 4).

Em 2004, foi criada a primeira linguagem de programação quântica funcional a QFC (Quantum Flow Charts), apresentada por Selinger. Ela é feita por uma versão funcional de flowcharts, mas também apresenta uma versão baseada em texto que é a QPL (Quantum Programming Language). A QFC é baseada no modelo QRAM de simulação quântica ([JOSE; PIQUEIRA; LOPES, 2013](#)).

O tipo de linguagem mais utilizado na programação é o primeiro grupo, os das linguagens imperativas, por estarem estabelecidas por mais tempo dentro da teoria da computação e informação.

Da mesma maneira, existe uma gama bastante variada de linguagens de programação quânticas imperativas. Destacando a linguagem QCL, segundo [Jose, Piqueira e Lopes \(2013\)](#) a linguagem de programação QCL (Quantum Computing Language) de Bernhard Ömer, criada em 1998 foi a primeira linguagem de programação quântica. Criada a partir da linguagem C, e funciona com o Modelo QRAM, onde uma máquina clássica controla um dispositivo quântico. A QCL, contém uma sub-linguagem clássica e funções de auto nível de programação quântica, como por exemplo o gerenciamento da memória. Em 2000, Sanders e Zuliani criaram a linguagem qGCL (Quantum Guarded Command Language) baseada nos códigos Dijkstra. Betteli, Calarco e Serafini apresentaram uma linguagem quântica baseada no C++, em 2003. Já em 2007, Mlnarik apresenta uma linguagem

baseada em C chamada de LanQ (Language Quantum), tratando operações clássicas e quânticas em paralelo (JOSE; PIQUEIRA; LOPES, 2013).

Existem ainda muitas outras linguagens de programação clássicas e quânticas que não foram mencionadas, o objetivo dessa seção era apenas dar um panorama geral das distintas linguagens de programação.

3.2 LINGUAGEM DE PROGRAMAÇÃO QUÂNTICA PARA COMPUTADORES CLÁSSICOS

Uma questão interessante sobre as linguagens de programação quânticas é: como pode-se utilizar delas sem se ter acesso a um Computador Quântico? Independentemente da linguagem utilizada, imperativa ou funcional, elas podem ser utilizadas ou implementadas em um computador clássico. De forma muito simplificada, o computador quântico é emulado no Computador Clássico como um jogo de Video Game, ou seja, o computador clássico tem a potencialidade de emular o computador quântico como emula um Video Game ¹.

Existem três formas que se destacam para emular um computador quântico. São eles: O Modelo de circuito quântico, Modelo da Máquina de Turing quântica e o Modelo QRAM (Quantum Random Access Machine). O primeiro se faz através de portas lógicas quânticas como os circuitos clássicos. O segundo faz-se através de transformações unitárias e medidas nunca são realizadas, já o terceiro permite que medidas e transformações unitárias sejam livremente intercaladas.

Segundo Jose, Piqueira e Lopes (2013), o modelo QRAM (Quantum Random Access Machine) pode ser explicado da seguinte maneira:

[...] diferentemente do modelo de circuito quântico, o modelo QRAM permite transformações unitárias e medidas serem livremente intercaladas. No modelo QRAM, um dispositivo quântico é controlado por um computador clássico. Esse dispositivo possui um enorme (mas finito) número de qubits individuais e endereçáveis, muito parecido com a memória RAM (Random Access Memory). O computador clássico envia uma sequência de instruções, às quais pode-se “aplicar uma transformação unitária U com os qubits i e j ” ou “medir o qubit i ”. O dispositivo quântico executa essas instruções e disponibiliza os resultados. Esse modelo foi construído com a premissa de que o computador quântico real será, de fato, um computador clássico com acesso a componentes da computação quântica. Diversas linguagens de programação quânticas usam o modelo QRAM (p. 4).

No modelo QRAM a máquina funcionaria como um computador híbrido, ou seja, apresenta funcionamento quântico e clássico. Na verdade teríamos um computador clás-

¹ Obviamente este exemplo não tem nenhum compromisso de se parecer, em termos de programação, com a utilização do computador clássico para “rodar” algoritmos quânticos.

sico que controla um dispositivo quântico. As instruções seriam dadas ao computador clássico que, quando necessário, envia as informações ao dispositivo quântico que realiza as transformações unitárias e retorna um valor ao computador, como exemplificado na figura 1.

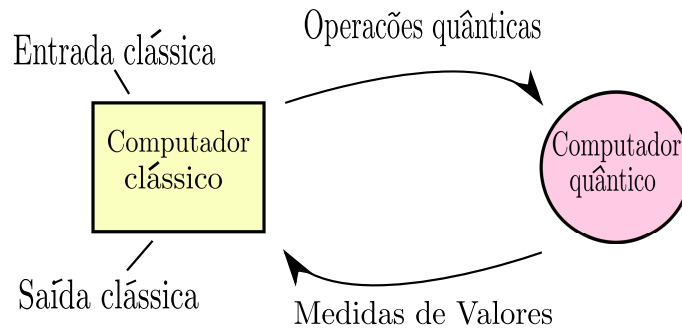


Figura 1 – Sistema como a linguagem híbrida.

Como foi citado na seção 2.2, a evolução de um sistema quântico pode ser representado pela aplicação de operadores unitários sobre o sistema. O computador clássico é uma calculadora, que pode ser programado para “criar” qualquer estado matemático (neste caso físico) que convier ao programador. Para se programar quanticamente um computador clássico, basta fazer com que o estado desejado seja submetido a operações quânticas.

3.3 LINGUAGEM DE PROGRAMAÇÃO QUÂNTICA QCL

Os precedimentos de como instalar um compilador QCL em um computador com o sistema operacional Linux UBUNTU 14.04 de 64 bits está descrito no apêndice A.1. Para começar a programar em QCL, após os procedimentos de instalação, deve-se abrir um terminal ou prompt de comando e digitar **qcl**, então pode-se iniciar a programação.

Se for da preferência, a programação da QCL pode ser feita por scripts de comando. Para isso basta abrir algum editor de texto, como gedit, e escrever todo o programa nele, em seguida basta salvar este arquivo em algum diretório utilizando a extensão **.qcl**. Com arquivo salvo, basta navegar pelas pastas do computador, pelo terminal, até encontrar aquela em que o programa foi salvo, para compilar o programa basta digitar no terminal **qcl nome do arquivo.qcl**. Se o programa foi construído corretamente, os resultados de sua implementação aparecerão no terminal. Na figura 2 pode-se verificar um exemplo de script QCL depois pronto.

Observa-se que o script da figura 2, conta com alguns comandos que serão melhor explicados adiante, e frases logo após duas barras direitas (**//**). Essas barras são usadas para fazer comentários, elas comentam toda a linha após a sua utilização, esses comentários não serão interpretados pelo compilador QCL como comandos, e sim vai ignorá-los,

mas o que vem antes de `//` será “lido”. Os comentários são úteis na utilização de scripts, não fazem muito sentido na programação direta pela linha de comando.

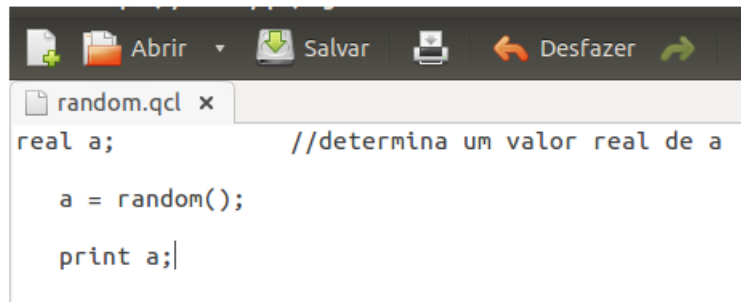


Figura 2 – Exemplo de script do QCL

O que foi mencionado acima são as formas conhecidas de escrever um programa, os conceitos e comandos para tal, serão descritos a partir de agora. De antemão ressalta-se que as mencionadas descrições estão presentes em [Ömer \(2000\)](#).

A linguagem de programação quântica QCL tem funcionamento análogo a programação em C. Tornando seu entendimento mais direto a quem tem um conhecimento pré estabelecido de C.

A linguagem QCL funciona na lógica de definição de variáveis e durante a execução do programa essas variáveis mudam seus valores iniciais, ou ganham valores. Essas variáveis devem ser definidas de acordo com a natureza dos valores que elas irão assumir, essa natureza pode ser de números inteiros, reais ou complexos, como ilustrado na tabela 2.

Tabela 2 – Definição das variáveis

Tipo	Descrição	Exemplos
int	Números Inteiros	1,2,3,...
real	Números Reais	1.2, 2.5, 3.7
complex	Números Complexos	(0,1), (-1,5), (5,-8)
boolean	Valores lógicos	Verdadeiro e falso
string	Frases	“Olá Mundo!”

Um exemplo aplicado de definição de variáveis pode ser representado pelo exemplo abaixo:

```
int i=1;
int j=5;
int k=i+j;
```


No mencionado exemplo, o programa soma os valores atribuídos a i e j , e o resultado dessa soma será denotado a contante k . Tem-se exemplos de comandos usados para determinar variáveis complexas e variáveis reais:

```
real i=pi;
complex z=(0,-1);
```

E neste caso obtêm-se os valores de $\pi = 3,14 \dots$ e o valor do número complexo z , e a sua sintaxe de definição é $z=(\text{Re},\text{Im})$, onde Re significa a parte real de z e Im a parte imaginária. É importante ressaltar que em ambos os exemplos existem conceitos que não foram mencionados, como soma e a constante π . Existem algumas operações que já estão definidas na própria linguagem, dentre elas está a soma(+), subtração (-), divisão (/) e multiplicação (*). Uma lista completa de operadores que estão previamente definidos em QCL está presente no anexo A. Existem constantes importantes, ou úteis que também já estão definidas na linguagem, o π é uma delas.

O comando *print* é usado para imprimir os valores associados as variáveis definidas no programa. E a sintaxe de como esses valores são mostrados segue a lógica ilustrada no exemplo abaixo. Quando o comando *print* é colocado, algum valor será mostrado no terminal, pode-se identificar quando esses valores aparecem quando observa-se dois pontos no terminal, em seguida aparece o valor desejado. Pode-se utilizar o comando *print* com diferentes sintaxes. Para imprimir o valor de várias variáveis é só separa-las por vírgula, logo após o próprio comando *print*, como no exemplo:

```
real i=pi;
complex z=(0,-1);
print i,z;
: 3,14159 (0,-1)
```

Agora, se o intuito é identificar as variáveis que o programa vai imprimir basta utilizar a seguinte sintaxe:

```
real i=pi;
print "O valor de pi é:",i;
:O valor de pi é: 3,14159
```

Dessa maneira, a mensagem que aparece entre aspas no programa será impressa antes do valor da variável, isso serve para quem está utilizando o programa, para que possa identificar melhor os valores que está obtendo.

Quanto a sintaxe de entrada dos valores associados as constantes, ela segue o mesmo formato que é apresentado na tabela 2. Números complexos são representados

por uma parte real e uma imaginária, como mencionado, números não inteiros devem ser representados como [1.5] ².

Na maioria das linguagens de programação, podemos definir alguns valores de entrada no programa durante a sua execução. Na QCL, isso pode ser feito usando o comando *input*, da seguinte maneira:

```
real i;  
input i;  
? real i? 1.34  
print i;  
: 1.34
```

De maneira básica o comando *input* tem duas formas de ser usado, a primeira está representada no exemplo acima. Onde deve-se definir uma variável, no mencionado caso a letra escolhida foi “*i*” e ela pode assumir uma entrada real, ou seja, permite números fracionários como valores de entrada. Em seguida da-se o comando “*input i*”, quando o programa for executado, nesse ponto ele irá solicitar a pessoa que o está executando um valor, e reconhece-se isso pela mensagem que é imprimida no terminal ou prompt de comando, que é como está ilustrado na terceira linha do programa acima. Basicamente, todas as solicitações de entrada de programas terão essa sintaxe, onde o tipo de variável que poderá ser inserido aparece entre os pontos de interrogação, e como no mencionado exemplo, deve-se dar um valor para que o programa prossiga (neste caso 1.34). Existe outra maneira de utilizar o comando *input*, essa maneira permite ao programador fazer uma descrição do que deseja-se colocar como valor de entrada, a sintaxe de utilização segue a mesma que o comando *print*, como podemos observar:

```
real i;  
input “Entre com um valor real:”i;  
? Entre com um valor real: 1.34  
print i;  
: 1.34
```

Neste caso, como temos a impressão da mensagem, não aparece a interrogação dos dois lados, mas sim, apenas no começo, mas o funcionamento do comando é o mesmo.

Ainda na tabela 2, apresenta-se uma entrada para valores lógicos, definida pela entrada *boolean*, e um exemplo e esse tipo de entrada de variáveis é.

```
boolean b;  
print b;  
: false
```

² Os números fracionários devem ser utilizados com ponto e não com vírgula.

Na linguagem QCL, estão definidas previamente várias funções, dentre elas as funções trigonométricas básicas. Uma maior descrição de quais funções estão definidas nesta linguagem podem ser encontradas no anexo B. Uma situação que as funções trigonométricas podem ser usadas é a seguinte.

```
real i=pi;
print sin(i), cos(i), tan(i);
: 0      -1      0
```

Na tabela 3, temos algumas operações importantes do QCL, elas podem ser bastante convenientes dependendo do programa que está sendo criado.

Tabela 3 – Tabela com demais funções importantes em QCL

Função	Descrição
$\text{ceil}(x)$	Arredonda o valor de x para cima
$\text{floor}(x)$	Arredonda o valor de x para baixo
$\text{max}(x, \dots)$	Máximo
$\text{min}(x, \dots)$	Mínimo
$\text{gcd}(x)$	Maior divisor comum
$\text{lcm}(x)$	Menor multiplicador comum
$\text{random}()$	Dá valores aleatórios de $[0,1)$

Um exemplo de programa que se encaixa nessas condições pode ser.

```
real i=pi;
print floor(i);
print ceil(i);
: 3
: 4
```

Dentro de qualquer programa os loopings são muito importantes, basicamente a definição de um looping em QCL, seguem uma sintaxe parecida com os loopings da linguagem C.

```
int i;
for i=0 to 2 { print i^2 }
: 0
: 1
: 4
```

Como a maioria das linguagens de programação, a QCL pode ter um programa escrito de várias formas. O programador pode escrever o programa da forma que convier mais a ele naquele determinado momento. Uma das formas mais utilizadas para e

programar é a definição de funções que antecedem o próprio programa. Para o programa não ficar muito complexo, ou muito “misturado”, é comum fazê-lo em partes. Cada parte do programa pode ser denominado como sendo uma função, e esta por sua vez pode ser chamada no meio do programa. Por conveniência pode-se preferir criar uma função que aplique o operador Hadamard em um determinado estado, para fazer isso basta seguir a seguinte sintaxe:

```
procedure hada(int n) {
  qureg q[n];
  H(q);
}
```

Dando o comando “hada” dentro do programa ele vai realizar a operação acima, mas se ele não for chamado durante o programa, essa parte do código simplesmente será ignorada. Nas funções, existe uma hierarquia entre os comandos iniciais para inicializa-las. Elas podem ser usadas de acordo com a natureza dos comandos que serão atribuídos a ela. O “*procedure*”, é o comando de mais alta hierarquia, os demais podem ser definidos “dentro” dele, que são: *operator*, *qufunct* e *functions*, citados em ordem de hierarquia. Basicamente, os de hierarquia mais baixa podem ser definidos dentro daqueles que tem uma hierarquia mais alta. Cada um é designado para uma função distinta dentro do programa, o *procedure*, é usado geralmente para escrever o próprio programa. O *operator* é utilizado para definição de operadores no programa, *qufunct* é usado para definir funções quânticas. Já *functions* é usado para definir funções clássicas no programa.

Em programação não existe um conceito mais ou menos importante, todos tem um papel a cumprir dentro de um programa, mas uma definição bastante importante é como alocar q-bits no estado da máquina, neste caso, quântica. Isso pode ser feito de maneira bem simples. Para alocá-se um estado com 4 q-bits de registro basta colocar o comando “*qureg a[4];*”. Então o estado “a” terá 4 q-bits alocados. No exemplo abaixo, foram alocados n q-bits no estado “q”, e isso pode ser feito, desde que quando o estado em questão for utilizado ele ganhe um número de q-bits, ou pelo programa ou pelo input do programador.

```
qureg q[n];
input “Números de q-bits que será utilizado:”,n;
?Número de q-bits que será utilizado: 10
```

Neste ponto é interessante ressaltar uma definição interessante, os q-bits de entrada de um programa em QCL sempre iniciam no estado $|0\rangle$, e funções lógicas como o *boolean* mencionado a alguns exemplos atrás sempre assumem o valor *false*, se nenhum valor é associado a eles pelo programador ou pelo programa. Mesmo assim, quando necessita-se que um estado seja iniciado em $|0\rangle$, usa-se o comando *reset* que leva todos os estados

para o estado $|0\rangle$, é uma forma de garantir que a inicialização do programa se inicie no estado desejado (ÖMER, 2000).

Existem ainda muitas outras definições e comandos para a QCL, mas os comandos básicos para a montagem de um programa foram citados e explicados. À medida que outros comandos forem citados, serão explicados devidamente. Por exemplo, os operadores quânticos não foram mencionados nesta seção, mas serão explicados na seção 4.2.

4 COMPUTAÇÃO QUÂNTICA

A ideia de Computador Quântico surgiu em 1982, quando independentemente os pesquisadores Richard Feynman e Paul Benioff apresentaram alguns dos resultados que eles acabaram chegando. Feynman chegou as suas conclusões através das simulações de sistemas quânticos em computadores clássicos. Benioff apontava que a miniaturização dos circuitos integrados apresentavam efeitos espúrios relacionados com a Mecânica Quântica, segundo ele esses efeitos poderiam ser aproveitados, o que culminaria na solução do problema de miniaturização dos circuitos e reparametrizando a computação de uma forma geral (JOSE; PIQUEIRA; LOPES, 2013).

Inegavelmente, as teorias da computação quântica e da informação quântica, são novas áreas de estudo da ciência, popularizando-se rapidamente e ganhando muitos olhares nesta década. Uma das vantagens de se construir um computador quântico, é a possibilidade de resolver problemas muito difíceis de serem resolvidos por computadores clássicos. Tais como: a fatoração por número inteiro; oferece uma nova criptografia, mais adequada aos recentes avanços na informação e computação quântica (AL-DAOUD, 2007).

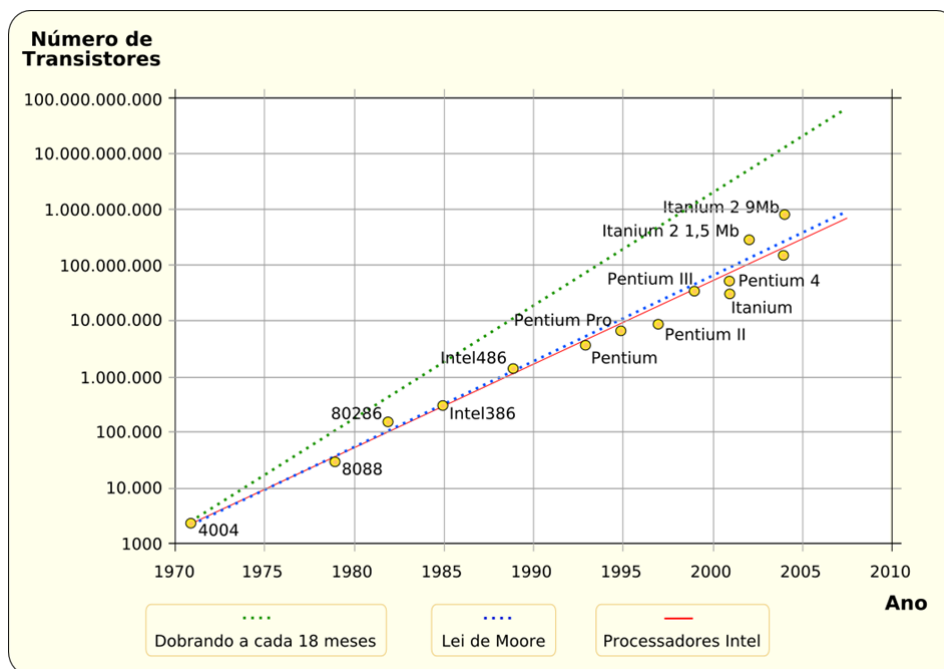
Qualquer computador, seja ele clássico ou quântico, segue uma mesma lógica em seu funcionamento. Basicamente o computador é um dispositivo que faz uso das seguintes ferramentas para funcionar (ÖMER, 2000).

1. Contem um estado físico que ganha o nome de estado da máquina;
2. É capaz de receber instruções **I** e transforma-las em estado da máquina;
3. Fornece meios para medir o estado da máquina, se interpreta um determinado valor como símbolo dos estados computacionais;

É interessante ressaltar um ponto no estudo da computação. Existe um postulado que deixa muito claro a questão da linearidade da construção do conhecimento em uma determinada área. Esse postulado ganhou o status de lei, embora não passasse apenas de um postulado com base na observação, e de certa forma ele funcionou mais como uma meta a ser alcançada do que uma lei de fato. Esse postulado é a Lei de Moore, ela foi descrita por Gordon Moore em 1962. Influenciou as duas maiores construtoras de processadores do mundo (INTEL e AMD) a seguirem o seu enunciado como meta, que é o seguinte: “Para um custo e um tamanho constante a quantidade de transistores em processador dobra a cada dezoito meses”. Para quem entende um pouco de teoria da computação e da informação sabe que, os computadores só são possíveis graças a tecnologia de transistores, e também que quanto maior o número deles, maior a capacidade de processamento do

computador. A capacidade de processamento interfere na velocidade do computador, pois a base de funcionamento do computador são os cálculos que ele pode realizar em um determinado intervalo de tempo, ou seja, um computador funciona a base de cálculos bem determinados, quanto maior a quantidade de transistores, mais cálculos por segundo o computador faz, e com isso se consegue uma máquina melhor. Sempre levando em consideração que outros fatores interferem na velocidade de uma máquina programável.

Figura 3 – Gráfico da Lei de Moore (PINTO et al., 2015)



Podemos observar na figura 3, que representa o aumento no número de transistores nos processadores construídos pela Intel, que esse comportamento linear tem sido obedecido. Existem divergências no que diz respeito a esse postulado, pois alguns colocam que a Lei de Moore será válida somente até 2020. Nesse ponto é importante destacar que isso depende de outros fatores. Se os cientistas conseguirem a façanha de diminuir as dimensões de um transistor, sem que os efeitos quânticos interfira no sistema, então a Lei de Moore poderá sobreviver pois mais algum tempo. Caso o contrário, não será mais considerada relevante, uma vez que não será mais possível dobrar o número de transistores em um mesmo tamanho de processador, como esse postulado nos diz.

4.1 MÁQUINA DE TURING

A criação do modelo teórico conhecido como Máquina de Turing foi proposta por Alan Turing e Alonzo Church. Inicialmente eles queriam responder a uma questão feita por David Hilbert, da forma como será explicado na seção 4.3. Em um de seus trabalhos Turing faz um questionamento, ele faz uma discussão em torno de as máquinas possuírem

inteligência ou não. Segundo ele, isso depende das definições de inteligência e o que é máquina, enfim, seus estudos culminaram na máquina de Turing e segundo os próprios Turing e Church ela tem quatro elementos principais (TURING, 1950):

1. Um programa: Como um computador comum;
2. Uma unidade de controle de estados finitos (processador);
3. Uma fita, como memória;
4. Uma cabeça de leitura e gravação;

O programa funciona como um programa qualquer de um computador. A unidade de controle funciona como o processador de um computador, onde pode ser considerado um conjunto de valores na memória $\langle q_1, \dots, q_m \rangle$, onde a matemática utilizada é a binária, ou seja, os valores que q_1 até q_m são de 0 ou 1. Onde m pode variar de forma considerável sem afetar a funcionalidade da máquina de Turing, então podemos considerar uma constante definida na unidade de controle, ela é também uma espécie de memória temporária. Existem ainda mais dois estados inteiros rotulados q_s e q_h , que são os estados de partida e o estado de parada respectivamente.

A fita da máquina de Turing é um objeto unidimensional, capaz de se estender indefinidamente em uma direção, dependendo da máquina de Turing que estamos considerando. Essa fita é dividida em células que são numeradas da forma: $0, 1, 2, 3, \dots$. Os valores que as células podem assumir dependem de um alfabeto denominado Γ que tem apenas quatro valores, que são: $0, 1, b$ (branco) e \triangleright , que marca a extremidade esquerda da fita. A cabeça de leitura só pode ler um valor de cada vez, o da célula o qual ele está em cima.

4.2 ESTADOS DE UM QUBIT

Na computação clássica, a sua unidade mais básica é o bit (1 e 0). A definição física do bit na computação clássica é determinada por dois estados contrários, que podem ser: passar corrente; não passar corrente, o spin de determinadas partículas estar magnetizado; não estar magnetizado. Para se representar letras em binário, é bem simples, associa-se as letras valores numéricos, por exemplo, pode-se numerar o alfabeto de 0 a 26, e cada um desses valores vai representar uma letra. A relação entre os números decimais e os números binários já um pouco mais direta (BULNES, 2005).

Como a computação clássica, a computação quântica também apresenta a sua unidade básica computacional, o quantum bit ou q-bit. O que é um q-bit? Segundo Nielsen e Chuang (2003), é um objeto matemático que pode ser associado a alguns sistemas

quânticos. E para não ficar preso em sistemas quânticos específicos, eles ganham um tratamento meramente matemático, em princípio. Diferentemente do bit clássico que pode assumir apenas dois estados puros, o q-bit pode assumir infinitos estados, caso não seja medido. Sistemas aos quais os q-bits podem estar associados são as diferentes polarizações de um fóton, e o spin $1/2$ de uma determinada partícula em um campo magnético.

Os dois autoestados que um q-bit pode assumir podem ser representados por

$$|0\rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \quad |1\rangle = \begin{pmatrix} 0 \\ 1 \end{pmatrix}. \quad (4.1)$$

O conjunto $\{|0\rangle, |1\rangle\}$, forma uma base no espaço de Hilbert, associada ao q-bit, que acaba ganhado o nome de base computacional (BULNES, 2005).

O estudo da computação quântica começa com estudo da aplicação do sistema quântico mais simples, um único q-bit. Um q-bit é um vetor ψ que pode ser descrito como $a|0\rangle + b|1\rangle$, onde a e b são dois números complexos que obedecem a relação $|a|^2 + |b|^2 = 1$. Um q-bit no estado $a|0\rangle + b|1\rangle$ pode ser visualizado como um ponto (θ, φ) , sobre uma esfera de raio unitário, conforme a figura 4 com $a = \cos(\frac{\theta}{2})$, $b = e^{i\varphi} \sin(\frac{\theta}{2})$. É interessante ressaltar que, qualquer estado que o q-bit venha a ocupar durante a programação, tem representação na esfera de Bloch.

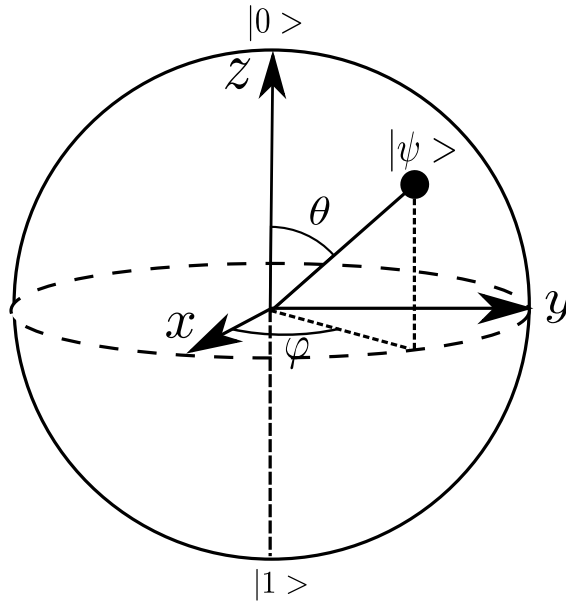


Figura 4 – Representação de um q-bit na esfera de Bloch

A esfera de Bloch, é um recurso interessante para se representar um q-bit e testarmos a suas propriedades. Nessa representação o q-bit $|\psi\rangle$ pode ser representado como:

$$|\psi\rangle = \cos\frac{\theta}{2}|0\rangle + e^{i\varphi}\sin\frac{\theta}{2}|1\rangle \quad (4.2)$$

Para representar um estado sobre o plano xy , o ângulo θ deve ser igual a 90° . Dessa forma:

$$|\psi_{xy}\rangle = \frac{\sqrt{2}}{2}|0\rangle + e^{i\varphi}\frac{\sqrt{2}}{2}|1\rangle. \quad (4.3)$$

No plano zx $\varphi = 0$, então:

$$|\psi_{zx}\rangle = \cos\frac{\theta}{2}|0\rangle + \sin\frac{\theta}{2}|1\rangle. \quad (4.4)$$

Para o plano zy $\varphi = 90$, e utilizando-se da Fórmula de Euler

$$e^{ib} = \cos(b) + i\sin(b), \quad (4.5)$$

fica:

$$|\psi_{zy}\rangle = \cos\frac{\theta}{2}|0\rangle + i\sin\frac{\theta}{2}|1\rangle. \quad (4.6)$$

Existe uma limitação nessa representação pela esfera de Bloch, não existe uma representação simples para dois q-bits ou mais, então a sua utilidade fica restrita a apenas um q-bit.

Qualquer tipo de operação que eventualmente seja feita sobre um q-bit devem preservar a sua norma, então essas operações são descritas por matrizes unitárias 2×2 . Um exemplo são as matrizes de Pauli

$$X \equiv \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}; \quad Y \equiv \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix}; \quad Z \equiv \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}.$$

As matrizes de Pauli também são conhecidas como portas lógicas, elas tem a finalidade de produzir influenciar sobre os estados de um ou mais q-bits. A porta X é conhecida como porta Not, pois a ação dela é levar um q-bit $|0\rangle$ para $|1\rangle$ e o q-bit $|1\rangle$ para $|0\rangle$ (NIELSEN; CHUANG, 2003). A sua ação é simples e pode ser descrita assim

$$X|0\rangle = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \cdot \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \end{bmatrix} = |1\rangle, \quad (4.7)$$

ou

$$X|1\rangle = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \cdot \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \end{bmatrix} = |0\rangle. \quad (4.8)$$

Um exemplo de programa de aplicação da porta X em QCL:

```
qreg q[1];          // o estado inicial é |0>
Not(q);             // aplica o operador Not sobre q
dump q;             // o estado de q = |1> ,
print "Esse é o valor de q depois da aplicação de Not:",q;
}
```

onde o comando “*Not(q)*” refere-se a porta X e o comando “*dump*” mostra o estado da máquina naquele instante.

A matriz de Pauli Y , tem uma aplicação um pouco distinta da porta X . Dada um q-bit genérico da forma $|\psi\rangle = |0\rangle + |1\rangle$, pode-se aplicar o operador da seguinte maneira:

$$Y|\psi\rangle = \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix} \cdot \left[\begin{pmatrix} 1 \\ 0 \end{pmatrix} + \begin{pmatrix} 0 \\ 1 \end{pmatrix} \right] = \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \end{bmatrix} = i(|1\rangle - |0\rangle) \quad (4.9)$$

Já o operador Z , causa uma mudança de fase em um q-bit $|\psi\rangle$, se $|\psi\rangle = |1\rangle + |0\rangle$ uma mudança de fase se caracteriza pela alteração do sinal de $+$ que está presente neste estado para o sinal de $-$, da seguinte maneira

$$Z|\psi\rangle = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \cdot \begin{bmatrix} 1 \\ 1 \end{bmatrix} = |0\rangle - |1\rangle, \quad (4.10)$$

um programa que representa o que está ilustrado acima, é:

```
operator fase(qureg q) {
  CPhase(pi,q);      //1/√2(-|0> -|1>)
}
procedure hada(int n=1) {
  qureg q[n];         //|0>
  H(q);               /* para poder aplicar essa operação sobre
q, primeiro deve-se definir a superposição de 1/√2(|0>
+|1>) que é feita por H, ou Mix.*a
fase(q);              //O estado da máquina vai para a 1 linha
do programa.
}
```

^a /* Texto */ é usado para comentar blocos de texto, não são restritos apenas uma linha

Onde *CPhase* é o comando de mudança de fase condicional, neste caso os q-bits de entrada terão uma alteração da fase em π , como mostrado na segunda linha do programa.

As portas lógicas, como já mencionado, realizam operações sobre os q-bits. Como já mencionado também, q-bits são estados físicos. Para aplicar operações sobre estes estados, deve-se ter outros entes físicos interferindo sobre eles. Na computação clássica, quem produz a interferência sobre os bits (também físicos), são os circuitos. De forma análoga, pode-se considerar as portas quânticas como circuitos quânticos. Os circuitos quânticos, em geral, tem ação nos estados aos quais estão associados, muito parecida com as ações de circuitos clássicos sobre os seus estados associados, mas diferentemente dos circuitos clássicos, os circuitos quânticos são reversíveis (CABRAL; LIMA; JR, 2004).

Outras três portas quânticas também tem importância significativa no estudo da computação quântica, que são as portas Hadamard (H), a porta de fase (S) e a porta $\frac{\pi}{8}$ (T)

$$H = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}; \quad S = \begin{bmatrix} 1 & 0 \\ 0 & i \end{bmatrix}; \quad T = \begin{bmatrix} 1 & 0 \\ 0 & \exp(i\frac{\pi}{4}) \end{bmatrix}.$$

A porta Hadamard, é utilizada para produzir a superposições de estados em um ou mais q-bits. Com a aplicação dessa porta todos os elementos do sistema passam a ter a mesma amplitude de probabilidade de serem medidos em uma eventual tomada de medidas. O hadamard tem um simples comando para ser utilizado em QCL (H). Um exemplo de implementação desta porta

```

procedure hada(int n) {
  qureg q[n];
  H(q);
}
```

O que interessa a programação é, como fazer uso de todos os estados que um q-bit pode assumir. A aplicação de portas ou de operadores serve para testar em que condições e como pode-se utilizar-se dessa propriedade do q-bit. Como o q-bit pode assumir valores intermediários de $|1\rangle$ e $|0\rangle$? Um primeiro estado a destacar-se é a aplicação do Hadamard que produz a superposição de estados de um q-bit. Após a aplicação do Hadamard, todos os estados possíveis de um q-bit estarão superpostos, isso pode ser representado por

$$H|0\rangle = \frac{|0\rangle + |1\rangle}{\sqrt{2}} \quad (4.11)$$

ou

$$H|1\rangle = \frac{|0\rangle - |1\rangle}{\sqrt{2}}. \quad (4.12)$$

O fato é, se realizar uma medida sobre qualquer um desses estados ele irá colapsar para $|0\rangle$ ou $|1\rangle$, mas tem-se a mesma probabilidade de medir um ou outro.

Seguindo a lógica da representação de Bloch, poderemos afirmar quase que de imediato que poderemos armazenar uma quantidade infinita de informação em apenas um q-bit, já que existem infinitos pontos em uma esfera de Bloch, mas quando ele é medido ele sempre irá colapsar para um único valor que pode ser $|1\rangle$ ou $|0\rangle$, ou seja, embora a matemática nos diz que é possível armazenar informação infinita em um q-bit, quando o medimos sempre vamos ter uma única informação.

Podemos ir além nessa discussão e então perguntarmos quanta informação pode ser armazenada em um q-bit se nós não o medirmos? E isso produz uma série de questionamentos que torna essa questão um tanto quanto delicada, pois como poderemos quantificar a informação sem medi-la? Quando temos um sistema de um q-bit evoluindo

de alguma maneira, ele sempre carrega consigo uma quantidade bastante grande do que se pode chamar de “informação oculta”, e essa informação cresce exponencialmente com a quantidade de q-bits envolvidos, e compreender essa informação quântica oculta é uma das mais interessantes fontes de estudo da computação quântica, e é ela que torna a mecânica quântica uma excelente ferramenta de processamento de informação.

4.3 ALGORITMOS

Na computação, é comum a ideia de algoritmo estar associada, e de fato é um conceito muito importante relacionado a teoria da computação e da informação. É importante ressaltar que a ideia de algoritmo não surgiu junto com os computadores, essa concepção é mais antiga, é de uma origem anterior a Cristo. Um exemplo bastante comum é o algoritmo de Euclides, para encontrar o máximo divisor comum (MDC), é um dos algoritmos mais antigos que se tem notícia e ele data de 300 a. C., mas foi apenas em 1930 que a moderna teoria dos algoritmos foi introduzida por Alonzo Church e Alan Turing, entre outros pioneiros da era dos computadores.

A moderna teoria dos algoritmos, foi formulada para tentar responder a uma questão levantada por David Hilbert no início do século XX. A questão era mais ou menos assim: “será possível existir um algoritmo capaz de responder a todos os problemas da matemática?” Esperava-se que a resposta fosse sim, mas ao final dos estudos descobriu-se que era não, mas essa questão impulsionou toda uma teoria, pois Church e Turing tiveram que capturar o significado intuitivo de algoritmo. Para fazer isso ambos estabeleceram a moderna teoria dos algoritmos. Isso culminou na criação de uma teoria, que dava a hipótese de construirmos uma máquina programável, que ficou conhecida como Máquina de Turing (MT) (seção 4.1) (NIELSEN; CHUANG, 2003).

Todo esse debate a respeito da MT resultou na teoria da computação moderna, bem como, dá suporte teórico para que se possa pensar em programar computadores quânticos, e embasa os estudos da teoria da informação quântica.

Para Al-Daoud (2007), quando fala-se em computação quântica, o que podemos perceber é que fica o questionamento, por que desenvolver uma computação quântica? Tem-se a promessa que dentro da mecânica quântica, de superar algoritmos clássicos, que poderão resolver problemas que a computação clássica é incapaz de responder, pelo simples fato que o computador clássico obedece a física clássica, e isso impõe muitas limitações no que o computador pode ou não fazer. E tudo isso depende da capacidade dos algoritmos que serão implementados nestes computadores quânticos.

Voltando o foco principal aos algoritmos, mas que é de fato um algoritmo? Segundo FERRARI e CECHINEL (2008), é uma sequência finita de passos para resolver um determinado problema. Sempre que se define um algoritmo, define-se um padrão, um

conjunto de passos a serem seguidos.

Ainda para [FERRARI e CECHINEL \(2008\)](#), existem alguns passos para elaboração de um bom algoritmo, e eles são:

1. Ações definidas de forma simples e sem ambiguidades;
2. As ações devem estar bem ordenadas;
3. Estabelecer as ações em um conjunto finito de passos;

e quando um algoritmo é executado por um computador ele tem ao menos 3 passos;

1. Entrada de dados;
2. Processamento de dados;
3. Saída de dados.

Independentemente do algoritmo ser quântico ou clássico, ele tem essa formulação. O que diverge em ambos os tipos de algoritmos é a natureza da operação que cada um irá realizar. Segundo [Oliveira \(2007\)](#), o conceito de algoritmo ainda é o mesmo de antes da computação existir, a diferença entre ambos é que um segue as leis da física clássica e o outro da física quântica. Na seção 4.2, abordou-se a questão dos circuitos quânticos ou portas lógicas quânticas; um algoritmo quântico é construído na perspectiva das portas quânticas, obedecendo os postulados mecânica quântica, bem como, apresentará os efeitos quânticos (descrições realizadas na seção 2.2).

Dentro dos algoritmos quânticos, tem-se dois algoritmos que se destacam. O primeiro é algoritmo baseado na transformada de Fourier quântica de Shor, e o segundo é o algoritmos de busca de Grover. Tem-se um terceiro algoritmo que também é bastante conhecido, mas de menor importância, que é o Algoritmo de Deutsch. O primeiro algoritmo quântico construído foi justamente o Algoritmo de Deutsch, seguido do algoritmo de Shor e depois o algoritmo de Grover. O algoritmo de Shor tem um ganho exponencial na velocidade de solução de problemas, se comparados aos seus análogos da computação clássica. O Algoritmo de Grover tem desempenho mais tímido, um ganho quadrático em relação aos seus análogos também, mas mesmo assim significativo ([AL-DAOUD, 2007](#)).

4.3.1 Algoritmo de Grover

Os algoritmos de busca, como o próprio nome já coloca, tem o objetivo de fazer uma busca. Esta pode ser feita em qualquer lista ou banco de dados. Os algoritmos de busca são bastante utilizados, principalmente na internet onde se encontram os sítios de busca, ou ferramentas de busca dos mais variados sítios.

Teoricamente, algoritmos de busca clássicos teriam um desempenho inferior aos algoritmos de busca quântica, denominados algoritmos de busca de Grover. Existem estudos onde a implementação do algoritmo de Grover já foi realizado com êxito em 1998, pois segundo [Chuang, Gershenfeld e Kubinec \(1998\)](#), o algoritmo de Grover foi implementado usando técnicas de ressonância magnética nuclear em uma solução de moléculas de clorofórmio, que foram utilizadas para implementar o Algoritmo de Grover para um sistema de quatro estados. Este estudo também foi, a primeira implementação de estados de um computador quântico.

A principal diferença entre algoritmos de busca clássicos e quânticos é a forma com que ambos fazem a busca em si. No algoritmo clássico a busca é feita de elemento por elemento, ou seja, se existe uma lista com N elementos, ao se fazer uma busca por um elemento em particular o algoritmo clássico poderá realizar N operações. Uma lista telefônica pode ser utilizada como exemplo, dado um determinado número de telefone, escreve-se um algoritmo para fazer a busca do número desejado. Se o algoritmo for clássico, ele vai comparar o número desejado com cada item da lista de telefone até encontrar o número por comparação. Diferentemente do algoritmo quântico, que ao invés de fazer a busca pelos números de telefone, que estão desordenados, ele irá fazer a busca pelos nomes que estão postos de forma ordenada, mais precisamente, em ordem alfabética. Isso implica que, em uma lista de N elementos o algoritmo quântico irá realizar \sqrt{N} operações para encontrar o elemento desejado, com alta probabilidade de acerto.

A atuação do algoritmo de busca de Grover pode ser definida no intervalo de 0 a $N - 1$, e por conveniência adota-se $N = 2^n$, com isso os índices podem ser representados por n bits. Isso implicará que o problema terá a possibilidade de M soluções, obedecendo a seguinte relação:

$$1 \leq M \leq N \quad (4.13)$$

Em uma aplicação do algoritmo de Grover com N elementos, o número de q-bits (n) utilizados, podem ser determinados pela seguinte igualdade

$$N = 2^n, \quad (4.14)$$

por simples operação

$$n = \log_2(N). \quad (4.15)$$

Para exemplificar: um problema de busca genérico que pode ser representado por uma função f , que recebe um inteiro x , no intervalo de 0 a $N - 1$. Para isso, define-se $f(x) = 1$ quando x é solução do problema e $f(x) = 0$ quando x não é solução do problema.

Para dar sequência ao exemplo, suponha-se também que tem-se um oráculo quântico, ele pode reconhecer a solução do problema da busca através de um q-bit-oráculo. O

oráculo é um operador unitário definido pela sua ação computacional:

$$O|x > |q > = |x > |q \oplus f(x) >, \quad (4.16)$$

onde $|x >$ é um estado de n q-bits, \oplus a soma de modulo 2, e $|q >$ o q-bit-oráculo. O oráculo soma uma função $f(x)$ ao q-bit-oráculo e o resultado dessa soma ficará mais claro adiante. $|x >$ é o estado de entrada dos itens onde a busca será realizada, e como citado, ele tem n q-bits associados a ele. Seguindo a equação 4.15, uma busca em uma lista com 5 elementos, serão necessários 3 q-bits.

Considere um exemplo em que o q-bit-oráculo é preparado no estado $|0 >$ e o operador oráculo atuando sobre o estado $|x > |0 >$. Após a aplicação do oráculo, obtêm-se um estado similar a equação 4.16, só mudando o q-bit-oráculo pelo estado $|0 >$, assim, $O|x > |0 > = |x > |0 \oplus f(x) >$. Nesse caso tem-se duas situações, uma quando $f(x) = 0$ e outra quando $f(x) = 1$, quando $f(x) = 0$ a equação $|x > |0 >$ permanecerá a mesma, ou seja, não se altera o estado da equação e o valor de $|x >$ não é solução, se $f(x) = 1$ o estado final é $|x > |1 >$ e então $|x >$ é solução.

Analizando também a atuação do operador Oráculo sobre o estado $|x > |1 >$, verifica-se que o estado final será $|x > |1 >$ quando $|x >$ não é solução, e será $|x > |0 >$ quando $|x >$ é solução. Então, percebe-se que o operador Oráculo não muda o estado de $|x >$ e muda o estado do q-bit Oráculo quando $|x >$ é solução.

Na implementação do algoritmo de Grover o q-bit-oráculo é preparado no estado $\frac{|0>-|1>}{\sqrt{2}}$, pode-se obtê-lo aplicando o operador Hadamard no estado $|1 >$. Dessa forma o estado inicial é escrito como

$$|x > \left(\frac{|0 > - |1 >}{\sqrt{2}} \right) = \frac{|x > |0 > - |x > |1 >}{\sqrt{2}}. \quad (4.17)$$

Pode-se observar que a aplicação do oráculo na equação 4.17 produz o resultado

$$O|x > \left(\frac{|0 > - |1 >}{\sqrt{2}} \right) = \begin{cases} -|x > \left(\frac{|0>-|1>}{\sqrt{2}} \right) & \text{se } f(x) = 1 \\ |x > \left(\frac{|0>-|1>}{\sqrt{2}} \right) & \text{se } f(x) = 0 \end{cases}$$

essas duas soluções podem ser resumidas na forma

$$O|x > \left(\frac{|0 > - |1 >}{\sqrt{2}} \right) = (-1)^{f(x)} |x > \left(\frac{|0 > - |1 >}{\sqrt{2}} \right) \quad (4.18)$$

observa-se após a atuação do operador Oráculo, ocorre uma mudança de fase no estado se $|x >$ é solução. Caso contrário o estado permanece o mesmo.

O algoritmo quântico de busca em si usa um único registro de entrada com n -qbits e consiste na aplicação sucessiva de operações conhecidas como iteração de Grover (G) conforme a figura 5. Cada vez que o operador (G) atua sobre o estado é realizado uma

consulta ao oráculo e o objetivo do algoritmo é realizar essa consulta o menor número de vezes possível com uma grande probabilidade de acerto. Para a implementação do algoritmo todos os n -q-bits do estado inicial são preparados no estado $|0\rangle$ que representamos dessa forma $|0\rangle^{\otimes n}$, então o operador Hadamard é aplicado para produzir a superposição de estados

$$|\psi\rangle = H^{\otimes n}|0\rangle, \quad (4.19)$$

onde $H^{\otimes n}$ é o operador Hadamard generalizado e quando atua sobre o estado $|0\rangle^{\otimes n}$ produz uma superposição de todos os estados possíveis do n -q-bits com mesma amplitude de probabilidade e com mesma fase. Esta a superposição ficará denotada pela relação

$$|\psi\rangle = \sum_{x=0}^{N-1} \frac{|x\rangle}{\sqrt{2^n}}. \quad (4.20)$$

Como vimos anteriormente, o valor de $2^n = N$, substituindo

$$|\psi\rangle = \frac{1}{N^{\frac{1}{2}}} \sum_{x=0}^{N-1} |x\rangle. \quad (4.21)$$

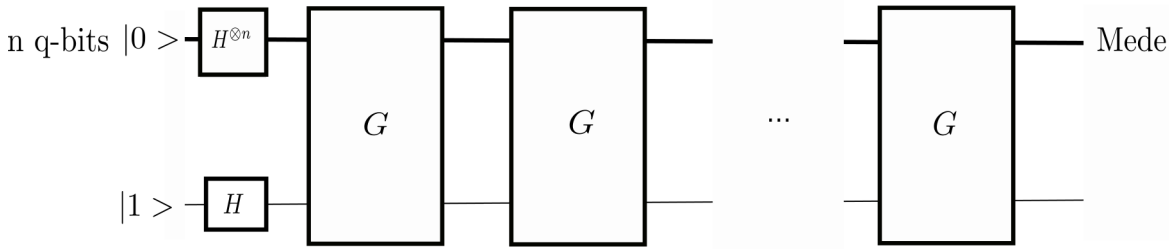


Figura 5 – Aplicação do algoritmo de Grover

O operador de Grover (G) pode ser representado esquematicamente como na figura 6 que consiste em quatro operações na seguinte sequência

1. Aplicação do oráculo O .
2. Aplicação do Hadamard generalizado.
3. Aplicação do deslocamento de fase condicional (DFC).
4. Aplicação do Hadamard generalizado.

A atuação do operador oráculo se dá conforme a equação 4.18 e o deslocamento de fase condicional gera uma mudança de fase em todos os estados do registro exceto para o estado $|0\rangle^{\otimes n}$. A constante de fase condicional pode ser implementada pela atuação do operador $(2|0\rangle^{\otimes n}\langle 0|^{\otimes n} - I)$ ¹. Desta forma a sequência de operações que descrevem o operador de Grover pode ser representado como

$$G = H^{\otimes n}(2|0\rangle^{\otimes n}\langle 0|^{\otimes n} - I)H^{\otimes n}O. \quad (4.22)$$

¹ I é operador identidade.

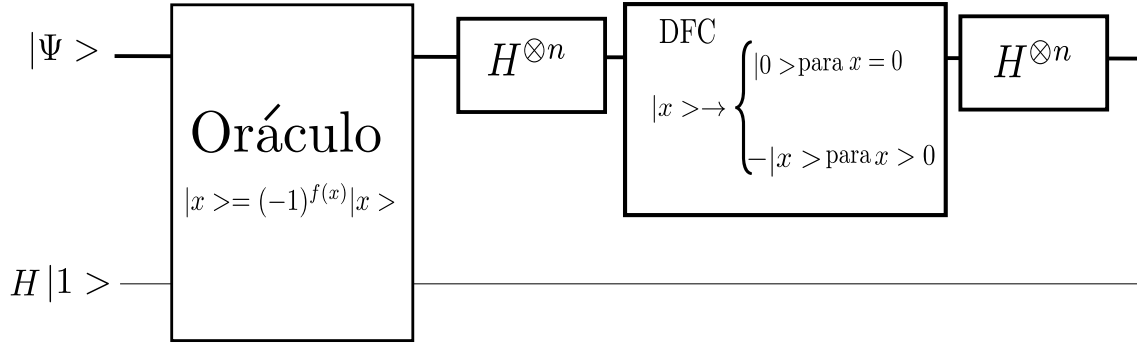


Figura 6 – Circuito para iteração de Grover

Vimos na equação 4.19 que o operador Hadamard generalizado atuando sobre o estado $|0\rangle^n$ gera uma superposição de todos os estados possíveis com mesma amplitude de probabilidade denotado por $|\psi\rangle$. Desta forma podemos reescrever 4.22 da seguinte maneira

$$G = (2|\psi\rangle\langle\psi| - I)O. \quad (4.23)$$

O estado ψ é formado por uma superposição de estados que a princípio pode conter estados são soluções do problema e também de estados que não são soluções do problema. Podemos portanto representar o estado ψ em um espaço bidimensional dado por

$$|\psi\rangle = \sqrt{\frac{N-M}{N}}|\alpha\rangle + \sqrt{\frac{M}{N}}|\beta\rangle, \quad (4.24)$$

onde $|\alpha\rangle$ representa a superposição dos estados que não são soluções e $|\beta\rangle$ representa a superposição dos estados que são soluções e que podem ser normalizados na forma a seguir

$$|\alpha\rangle \equiv \frac{1}{\sqrt{N-M}} \sum_x |x\rangle \quad (\text{para } x \text{ não solução})$$

$$|\beta\rangle \equiv \frac{1}{\sqrt{M}} \sum_x |x\rangle \quad (\text{para } x \text{ solução}).$$

Fazendo uma parametrização da forma $\sin\frac{\theta}{2} = \sqrt{\frac{M}{N}}$ pode-se reescrever a equação 4.24 da seguinte forma

$$|\psi\rangle = \cos\frac{\theta}{2}|\alpha\rangle + \sin\frac{\theta}{2}|\beta\rangle. \quad (4.25)$$

Aplicações sucessivas do operador de Grover (G) sobre a equação 4.25, produzirão uma maior probabilidade de se medir a resposta do problema. Podemos observar isso se pensarmos nos estados $|\alpha\rangle$ e $|\beta\rangle$ como sendo dois eixos ortogonais, tal qual a representação da figura 7. Na figura 7a representamos o estado $|\psi\rangle$ em termos de suas componentes $|\alpha\rangle$ e $|\beta\rangle$.

A função $|\psi\rangle$ é submetida a uma aplicação do operador Oráculo, e lembrando que a função $|\psi\rangle$ está separada como função de dois estados um que representa o conjunto

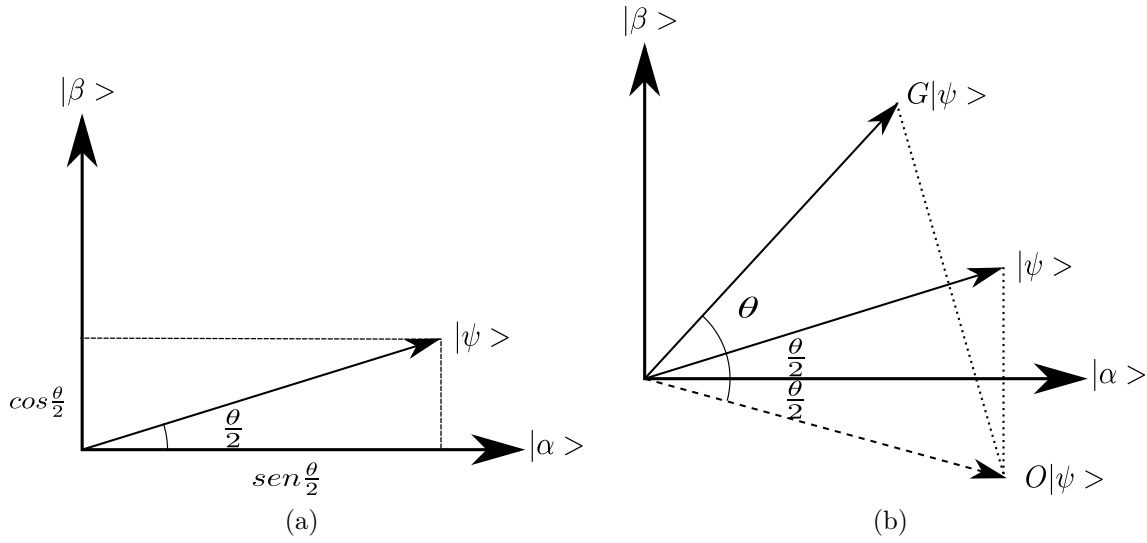


Figura 7 – Representação geométrica a implementação G .

de soluções e outro que representa o conjunto de não soluções, e portanto o resultado da aplicação do operador oráculo será uma mudança de fase no estado $|\beta\rangle$ que representa as soluções do problema, ou seja,

$$O|\psi\rangle = \cos\frac{\theta}{2}|\alpha\rangle - \sin\frac{\theta}{2}|\beta\rangle. \quad (4.26)$$

Podemos observar na figura 7b que a atuação do operador Oráculo sobre o estado $|\psi\rangle$ representa uma reflexão sobre o eixo $|\alpha\rangle$. Em seguida aplica-se o operador $(2|\psi\rangle\langle\psi| - I)$ sobre estado $O|\psi\rangle$ o qual gera o seguinte resultado

$$G|\psi\rangle = \cos\frac{3\theta}{2}|\alpha\rangle + \sin\frac{3\theta}{2}|\beta\rangle, \quad (4.27)$$

que é a atuação do operador de Grover no estado $|\psi\rangle$. Representamos o estado $G|\psi\rangle$ na figura 7b e observamos uma reflexão do estado $O|\psi\rangle$ em torno do vetor $|\psi\rangle$.

O algoritmo de Grover como representado na figura 5 requer uma aplicação sucessiva do operador de Grover e portanto depois de k operações temos o seguinte resultado

$$G^k|\psi\rangle = \cos[(k + \frac{1}{2})\theta]|\alpha\rangle + \sin[(k + \frac{1}{2})\theta]|\beta\rangle. \quad (4.28)$$

Observamos que aumentando o número de iterações k aumentamos a probabilidade de se medir as soluções, pois para k suficientemente grande teremos $\sin[(k + \frac{1}{2})\theta] \approx 1$, ou seja, $(\frac{2k+1}{2})\theta \approx \frac{\pi}{2}$. Então o número de iterações k necessárias para se medir as soluções com probabilidades suficientemente altas deve ser

$$k = \frac{\pi}{2\theta} + \frac{1}{2}. \quad (4.29)$$

Considerando um banco de dados com N suficientemente grande tal que $\sin \frac{\theta}{2} = \sqrt{\frac{M}{N}}$ possa ser aproximando para $\frac{\theta}{2}$. Então substituindo θ na equação 4.29 obteremos

$$k = \frac{\pi}{4} \sqrt{\frac{N}{M}} \quad (4.30)$$

A descrição do algoritmo de Grover acima, pode ser encontrada em [Nielsen e Chuang \(2003\)](#). Um exemplo de algoritmo de Grover escrito na linguagem QCL está apresentado abaixo. Esse exemplo foi retirado do próprio QCL, onde ele define os seus principais algoritmos e operadores.

```

1  qfunct query(qureg x,quvoid f,int n) {
2      int i;
3      for i=0 to #x-1 {
4          if not bit(n,i) { Not(x[i]); } }
5      CNot(f,x);
6      for i=0 to #x-1 {
7          if not bit(n,i) { !Not(x[i]); } } }
8
9  operator diffuse(qureg q) {
10     H(q);
11     Not(q);
12     CPhase(pi,q);
13     !Not(q);
14     !H(q); }
15
16  procedure grover(int n) { //Inicia o programa
17     int l=floor(log(n,2))+1; //calcula a quantidade de q-bits
18     int m=ceil(pi/8*sqrt(2^l)); //Calcula a quantidade de iterações
19     int x;
20     int j;
21     qureg q[l]; //Aloca um registro q com l q-bits
22     qureg f[1]; //Aloca um registro f com 1 q-bit que será o q-bit-oráculo
23  print l,"qubits, usando",m,"itarações"; } //Imprime os valores de l e m
24  reset; //Reseta os q-bits para que todos iniciem no estado 0
25  H(q); //Aplica a superposição de estados no registro q
26  for j= 1 to m { //Loping da quantidade de iterações
27      query(q,f,n); // calculate C(q)

```

```

28      CPhase(pi,f);           //Muda a fase em pi de f
29      !query(q,f,n); // calculate C(q)
30      diffuse(q); }
31      measure q,x;           //Mede o valor dos estados de q
32      print "valor medido:",x; //Encontra uma resposta e imprime
33      } until x==n;
34      reset; }

```

Neste exemplo optou-se por numerar as linhas do programa para que fique mais direta a localização comandos que estarão sendo citados.

O programa começa de fato na linha 16, onde o comando *procedure* aparece definindo-o. O nome do programa foi determinado como *grover*, como pode ser observado ainda na mesma linha. Além disso, uma variável inteira *n* é definida, essa variável *n* é o número que o algoritmo vai tentar buscar durante a execução do programa, ele é definido pelo programador antes do algoritmo “rodar” de fato. Para ter uma maior interatividade com o algoritmo pode-se escrever ao fim do script, depois da chave que fecha o programa, os seguintes comandos:

```

int n;
input“escreva o valor que deseja buscar:”,n;
grover(n);

```

com isso, o programa, antes de executar, solicitará um valor para ser utilizado para realizar a busca. Os valores que podem ser utilizados para realizar uma busca pelo Algoritmo de Grover, não devem ser muito grandes, uma vez que o compilador QCL tem 64 estados a serem ocupados, se o valor de entrada for muito alto, o número de estados pode ser extrapolado, impossibilitando a execução do algoritmo.

Como o algoritmo só permite a busca de valores numéricos, quando a mensagem

```
:escreva o valor que deseja buscar:
```

for impressa, apenas um valor inteiro será aceito como valor a ser procurado. Pode-se usar como exemplo o número inteiro 5 afim de que o programa realize a busca por este número. O algoritmo vai procurar o valor do estado que representa o 5, que neste caso será $|101\rangle$ na base computacional. Isso ocorre com qualquer outro valor de entrada.

Seguindo na explicação do algoritmo, logo após a entrada do valor a ser buscado, que como já foi usado pode ser 5. Na linha 16 do programa a variável *n* assumirá o valor de 5. Esse valor vai ser utilizado para calcular o número de q-bits necessários para realizar a busca na linha 17, onde, como foi visto o comando *floor* arredonda os valores associados a ele para baixo, e calcula-se o valor do $\log(n)$ na base 2. Se $n = 5$, o \log será

aproximadamente 2,32, o comando *floor* arredondará esse valor para 2 e soma-se 1 a esse valor, chegando a conclusão que são necessários 3 q-bits (3 será o valor de l) para realizar a busca por 5. Na linha 18, calcula-se o número de iterações necessárias para poder medir o estado desejado, isso é feito através de uma conta que está descrita na referida linha. O comando *ceil* é usado para arredondar os valores para cima, e substituindo o valor de 1 na equação chega-se a 2, ou seja, são necessárias duas iterações para medir o valor de 5.

Um sistema de 3 q-bits gera um espaço com 8 estados, pois o número de estados corresponde a 2^n , e como tem-se 3 q-bits no exemplo de busca pelo 5. Cada um desses estados podem ser numerados de 0 a 7, e o algoritmo de busca implementado para a busca 5 vai identificar o estado que corresponde ao 5º estado. Segue abaixo os estados numerados de 0 a 7 e seus respectivos valores na base computacional:

- 0 - $|000\rangle$
- 1 - $|001\rangle$
- 2 - $|010\rangle$
- 3 - $|011\rangle$
- 4 - $|100\rangle$
- 5 - $|101\rangle$
- 6 - $|110\rangle$
- 7 - $|111\rangle$

Na linha 21, define-se o valor de um q-bit de registro q com $l = 3$ q-bits. A linha 22 define o q-bit que será usado como q-bit-oráculo, portanto o estado da máquina é um estado de 4 q-bits. Ambos os registros quânticos iniciam no estado $|0000\rangle$, onde o primeiro 0, olhando da esquerda para a direita, representa o valor do q-bit-oráculo. Na linha 24, aplica-se o comando *reset* que leva o estado de todos os q-bits do sistema para $|0000\rangle$, caso eles não estejam.

Na linha 25, o operador Hadamard é aplicado nos q-bits do estado de registro q , essa aplicação produz o seguinte estado no exemplo de busca utilizado até o momento (busca do 5):

$$H|0000\rangle = \frac{1}{\sqrt{8}}|0000\rangle + \frac{1}{\sqrt{8}}|0001\rangle + \frac{1}{\sqrt{8}}|0010\rangle + \frac{1}{\sqrt{8}}|0011\rangle + \frac{1}{\sqrt{8}}|0100\rangle + \frac{1}{\sqrt{8}}|0101\rangle + \frac{1}{\sqrt{8}}|0110\rangle + \frac{1}{\sqrt{8}}|0111\rangle \quad (4.31)$$

coforme a equação 4.19.

Subsequentemente, na linha 26 da-se o início do primeiro loping, que se refere a quantidade de iterações que serão realizadas. Esse loping termina na linha 30, e será executado m vezes como o descrito na linha 26. No exemplo de busca de 5, como $m = 2$, esse loping é executado 2 vezes, da maneira que será explicado asseguir.

A primeira linha (37) do loping, faz a chamada de uma função quântica, o *query*. Essa função, tem operação computacional bastante interessante, e está definida a partir

da primeira linha do programa que está descrito acima. Em *query*, o estado da máquina (Θ) pode ser expresso, após ser negado pelo comando *CNot*:

$$\begin{aligned} \Theta = & \frac{1}{\sqrt{8}}|0000\rangle + \frac{1}{\sqrt{8}}|0001\rangle + \frac{1}{\sqrt{8}}|0010\rangle + \frac{1}{\sqrt{8}}|0011\rangle + \\ & \frac{1}{\sqrt{8}}|0100\rangle + \frac{1}{\sqrt{8}}|1101\rangle + \frac{1}{\sqrt{8}}|0110\rangle + \frac{1}{\sqrt{8}}|0111\rangle. \end{aligned} \quad (4.32)$$

O comando *Cphase*, muda a fase do q-bit oráculo em π

$$\begin{aligned} \Theta = & \frac{1}{\sqrt{8}}|0000\rangle + \frac{1}{\sqrt{8}}|0001\rangle + \frac{1}{\sqrt{8}}|0010\rangle + \frac{1}{\sqrt{8}}|0011\rangle + \\ & \frac{1}{\sqrt{8}}|0100\rangle - \frac{1}{\sqrt{8}}|1101\rangle + \frac{1}{\sqrt{8}}|0110\rangle + \frac{1}{\sqrt{8}}|0111\rangle \end{aligned} \quad (4.33)$$

o comando *!query* (linha 29), tem ação idêntica a *query*, e após a passagem por esse comando o estado da máquina fica

$$\begin{aligned} \Theta = & \frac{1}{\sqrt{8}}|0000\rangle + \frac{1}{\sqrt{8}}|0001\rangle + \frac{1}{\sqrt{8}}|0010\rangle + \frac{1}{\sqrt{8}}|0011\rangle + \\ & \frac{1}{\sqrt{8}}|0100\rangle + \frac{1}{\sqrt{8}}|0101\rangle + \frac{1}{\sqrt{8}}|0110\rangle - \frac{1}{\sqrt{8}}|1111\rangle. \end{aligned} \quad (4.34)$$

Ao término da ação de *!query*, o estado da máquina é:

$$\begin{aligned} \Theta = & \frac{1}{\sqrt{8}}|0000\rangle + \frac{1}{\sqrt{8}}|0001\rangle + \frac{1}{\sqrt{8}}|0010\rangle + \frac{1}{\sqrt{8}}|0011\rangle + \\ & \frac{1}{\sqrt{8}}|0100\rangle - \frac{1}{\sqrt{8}}|0101\rangle + \frac{1}{\sqrt{8}}|0110\rangle + \frac{1}{\sqrt{8}}|0111\rangle. \end{aligned} \quad (4.35)$$

A ação do *Diffuse* é bem simples, a aplicar o operador Hadamard o estado fica:

$$\begin{aligned} \Theta = & 0,75|0000\rangle + 0,25|0001\rangle - 0,25|0010\rangle + 0,25|0011\rangle + \\ & 0,25|0100\rangle - 0,25|0101\rangle + 0,25|0110\rangle - 0,25|0111\rangle, \end{aligned} \quad (4.36)$$

e depois da ação do operador *Not*

$$\begin{aligned} \Theta = & -0,25|0000\rangle + 0,25|0001\rangle - 0,25|0010\rangle + 0,25|0011\rangle + \\ & 0,25|0100\rangle - 0,25|0101\rangle + 0,25|0110\rangle + 0,75|0111\rangle, \end{aligned} \quad (4.37)$$

depois de uma nova aplicação de *CPhase*

$$\begin{aligned} \Theta = & -0,25|0000\rangle + 0,25|0001\rangle - 0,25|0010\rangle + 0,25|0011\rangle + \\ & 0,25|0100\rangle - 0,25|0101\rangle + 0,25|0110\rangle + 0,75|1111\rangle. \end{aligned} \quad (4.38)$$

A aplicação de *!Not* é análoga a *Not*, então:

$$\begin{aligned} \Theta = & -0,75|0000\rangle + 0,25|0001\rangle - 0,25|0010\rangle + 0,25|0011\rangle + \\ & 0,25|0100\rangle - 0,25|0101\rangle + 0,25|0110\rangle - 0,25|0111\rangle, \end{aligned} \quad (4.39)$$

em seguida a aplicação do operador *!H*, que tem mesma ação que *H*, logo

$$\begin{aligned} \Theta = & -0,17678|0000\rangle - 0,17678|0001\rangle - 0,17678|0010\rangle - 0,17678|0011\rangle - \\ & -0,17678|0100\rangle - 0,88388|0101\rangle - 0,17678|0110\rangle - 0,17678|0111\rangle. \end{aligned} \quad (4.40)$$

Tabela 4 – Números buscados (NINGTYAS; MUTIARA, 2010)

Valor de entrada	nº de q-bits	iterações	valor medido	total de iteração
10	4	2	10	2
30	5	3	30	3
175	8	7	175	7
500	9	9	175-500	18
1000	10	13	685-808-1000	39
1500	11	18	1500	18
2000	11	18	1224-1714-2000	54
8000	13	36	8000	36
10000	14	51	11287 - 10000	102

Tabela 5 – Busca do número 1500

Valor Medido	Total de Iterações
162 - 1500	36
800 - 1500	36
589 - 1500	36
637 - 1500	36
1500	18
1500	18
1500	18
1417 - 1500	36
1500	18
1197 - 1500	36

Enfim, o programa é finalizado, o comando *measure* é usado para medir o estado da máquina, o que estiver com a maior amplitude de probabilidade terá maiores chances de ser medido, mas pode ocorrer de ele não ser medido, se isso acontecer o programa repete a operação, isso é garantido pelo comando *until*.

Alguns exemplos de números buscados com esse algoritmo são os que aparecem na tabela 4.

Pode-se observar na tabela 4 que alguns valores foram medidos de forma insatisfatória então o programa repetiu a operação, obviamente que com essa ação o número de iterações necessária aumenta, como também é ilustrado na respectiva tabela. Afim de demonstrar o cunho probabilístico do algoritmo, buscou-se o número 1500 e o resultado desta busca está colocado na tabela 5.

5 CONSIDERAÇÕES FINAIS

Neste trabalho implementamos o algoritmo de Grover utilizando a QCL como uma linguagem quântica. O algoritmo de Grover prevê \sqrt{N} operações para encontrar a resposta enquanto que os algoritmos clássicos precisam, na pior das hipóteses, de N operações. Foram realizadas buscas por números variando de 5 a 10 000 e em alguns casos o algoritmo não encontrou o resultado correto na primeira medida, mas tendo sucesso na segunda ou terceira medida. Quando colocado para buscar o número 1500 obteve-se sucesso na primeira medida em 4 de dez simulações e nas outras 6 simulações o número buscado foi obtido na segunda medida com um total de 36 iterações. Mesmo quando o algoritmo de Grover não encontra o valor buscado na primeira medida, o número de operações ainda é muito menor do que o previsto em um computador clássico.

Realizamos a simulação de um computador quântico em um computador clássico utilizando uma linguagem de programação quântica a QCL. Com o surgimento dos computadores quânticos, os computadores clássicos continuarão existindo, pois a proposta da QCL é a utilização de um sistema híbrido de um computador clássico juntamente com um computador quântico.

Foi possível compreender a manipulação de um qbit através da esfera de Bloch e como estas manipulações são implementadas na QCL. Observamos que a QCL prepara o estado quântico inicial no estado $|0\rangle$ independente do número de qbits que integrem o estado inicial. A manipulação do estado inicial até o estado desejado se dá por meio de portas lógicas e neste sentido destacamos o operador Hadamard que é utilizado para gerar uma superposição de todos os estados possíveis, mas não gera emaranhamento. Estados emaranhados são utilizados nos algoritmos quânticos, inclusive no algoritmo de Grover que foi obtido com o comando CNot.

Referências

- AL-DAOUD, E. Quantum Computing for Solving a System of Nonlinear Equations over GF (q). *Int. Arab J. Inf. Technol.*, v. 4, n. 3, p. 201–205, 2007. Disponível em: <http://ccis2k.org/iajit/PDF/vol.4,no.3/3-AlDaoud.pdf>. Citado 4 vezes nas páginas 14, 45, 52 e 53.
- ALVES, F. L. Computação quântica: fundamentos físicos e perspectivas. maio 2015. Disponível em: <http://repositorio.ufla.br/jspui/handle/1/9369>. Citado na página 29.
- ALVES, F. L. Computação quântica: fundamentos físicos e perspectivas. 2015. Disponível em: <http://repositorio.ufla.br/handle/1/9369>. Citado na página 31.
- ANTON, H.; BUSBY, R. C. *Algebra Linear Contemporânea*. [S.l.]: Bookman Editora, 2006. ISBN 978-85-7780-091-9. Citado 2 vezes nas páginas 21 e 22.
- BASSALO, J. M. F. *Eletrodinâmica Quântica*. [S.l.]: Editora Livraria da Física, 2006. ISBN 978-85-88325-52-4. Citado na página 24.
- BOLDRINI, J. L. *Álgebra Linear*. Sao Paulo (SP): Harbra, 1986. ISBN 978-85-294-0202-4. Citado 2 vezes nas páginas 17 e 24.
- BULNES, J. J. D. *Emaranhamento e separabilidade de estados em Computação Quântica por Ressonância Magnética Nuclear*. Tese (Doutorado) — Tese (Doutorado)—Centro Brasileiro de Pesquisas Físicas, Rio de Janeiro, Setembro, 2005. Disponível em: http://www.cbpf.br/~fmelo/thesis/tese_Juan_qig_2005.pdf. Citado 4 vezes nas páginas 30, 31, 47 e 48.
- CABRAL, G. E. M.; LIMA, A. F. D.; JR, B. L. Interpretando o algoritmo de Deutsch no interferômetro de Mach-Zehnder. *Revista Brasileira de Ensino de Física*, v. 26, n. 2, p. 109–116, 2004. Disponível em: <http://www.scielo.br/pdf/rbef/v26n2/a05v26n2>. Citado na página 50.
- CHUANG, I. L.; GERSHENFELD, N.; KUBINEC, M. Experimental Implementation of Fast Quantum Searching. *Physical Review Letters*, v. 80, n. 15, p. 3408–3411, abr. 1998. Disponível em: <http://link.aps.org/doi/10.1103/PhysRevLett.80.3408>. Citado na página 54.
- CONCEITOS de Física Quântica 1. [S.l.]: Editora Livraria da Física, 2003. ISBN 978-85-88325-17-3. Citado na página 28.
- EVARISTO, J. *Programming in C Language: For Beginners*. [S.l.: s.n.], 2000. Citado 2 vezes nas páginas 33 e 34.
- FERRARI, F.; CECHINEL, C. Introdução a Algoritmos e Programação. *Bagé: Universidade Federal do Pampa, Campus Bagé*, 2008. Disponível em: <http://www.conexaoinformatica.net.br/apostilas/Eletronica/Algoritmos/FFerrari-CCechinel-Introducao-a-algoritmos.pdf>. Citado 2 vezes nas páginas 52 e 53.

- GAUSS, C. F. *Disquisitiones Arithmeticae...* [S.l.]: Nabu Press, 2012. ISBN 978-1-278-01394-7. Citado na página 19.
- GONCALVES, H. S.; others. A importância das matrizes e transformações lineares na computação gráfica. 2013. Disponível em: <<https://repositorio.bc.ufg.br/tede/handle/tde/2949>>. Citado 2 vezes nas páginas 17 e 19.
- GRECA, I. M.; MOREIRA, M. A. UMA REVISÃO DA LITERATURA SOBRE ESTUDOS RELATIVOS AO ENSINO DA MECÂNICA QUÂNTICA INTRODUTÓRIA (A review of the literature on studies regarding the teaching of introductory quantum mechanics). *Investigações em Ensino de Ciências*, v. 6, n. 1, p. 29–56, 2001. Disponível em: <http://www.educadores.diaadia.pr.gov.br/arquivos/File/2010/artigos_teses/fisica/artigos/revisao_literatura.pdf>. Citado na página 15.
- GRIFFITHS, D. J. *Mecânica Quântica*. São Paulo: Pearson, 2007. ISBN 978-85-7605-927-1. Citado 2 vezes nas páginas 26 e 27.
- JOSE, M. A.; PIQUEIRA, J. R. C.; LOPES, R. d. D. Introduction to quantum programming. *Revista Brasileira de Ensino de Física*, v. 35, n. 1, p. 1–9, mar. 2013. ISSN 1806-1117. Disponível em: <http://www.scielo.br/scielo.php?script=sci_abstract&pid=S1806-11172013000100006&lng=en&nrm=iso&tlng=pt>. Citado 6 vezes nas páginas 33, 34, 35, 36, 45 e 73.
- JR, O. P. Histórias contrafactuais: o surgimento da física quântica. *Estudos Avançados*, v. 14, n. 39, p. 175–204, 2000. Disponível em: <http://www.scielo.br/scielo.php?pid=S0103-40142000000200013&script=sci_arttext>. Citado na página 14.
- LIMA, T. C.; MIOTO, R. C. T. Procedimentos metodológicos na construção do conhecimento científico: a pesquisa bibliográfica. *Revista Katálisis*, v. 10, n. 1, p. 37–45, 2007. Disponível em: <<http://www.scielo.br/pdf/rk/v10nspe/a0410spe.pdf>>. Citado na página 13.
- MALAJOVICH, G. Álgebra Linear. *Disponível em*, v. 16, 2007. Disponível em: <<http://files.sistele7.webnode.com/200000337-b05eab1590/al2.pdf>>. Citado na página 18.
- MOORE, G. E.; others. Progress in digital integrated electronics. *IEDM Tech. Digest*, v. 11, 1975. Disponível em: <<http://www.lithoguru.com/scientist/CHE323/Moore1975.pdf>>. Citado na página 14.
- MORAIS, R. de. Ciência e tecnologia. *Revista Ciência e Tecnologia*, v. 8, n. 13, 2010. Disponível em: <<http://www.revista.unisal.br/sj/index.php/123/article/viewArticle/73>>. Citado na página 13.
- NEVES, V. Introdução à Teoria dos Números. *Departamento de Matemática da Universidade de Aveiro*, 2001. Disponível em: <<http://arquivoescolar.org/bitstream/arquivo-e/76/1/itn.pdf>>. Citado na página 19.
- NICHOLSON, W. K. *Álgebra Linear - 2.ed.* [S.l.]: AMGH Editora, 2015. ISBN 978-85-8055-477-9. Citado na página 18.
- NIELSEN, M. A.; CHUANG, I. L. *Computação quântica e informação quântica*. [S.l.]: Bookman, 2003. ISBN 978-85-363-0554-7. Citado 11 vezes nas páginas 14, 15, 18, 19, 20, 25, 27, 47, 49, 52 e 59.

- NINGTYAS, D. K.; MUTIARA, A. B. Simulating Grover's Quantum Search in a Classical Computer. *arXiv preprint arXiv:1003.1930*, 2010. Disponível em: <<http://arxiv.org/abs/1003.1930>>. Citado 2 vezes nas páginas 9 e 63.
- OLIVEIRA, N. A. A Utilização do Algoritmo Quântico de Busca em Problemas da Teoria da Informação. 2007. Disponível em: <http://docs.computacao.ufcg.edu.br/posgraduacao/dissertacoes/2007/Dissertacao_NiginiAbilioOliveira.pdf>. Citado 6 vezes nas páginas 14, 15, 25, 28, 29 e 53.
- OLIVEIRA, T. R. d.; OLIVEIRA, M. C. d. Tese de Doutorado, *Emaranhamento e estados de produto de matrizes em transições de fase quânticas*. 2008. Disponível em: <<http://www.bibliotecadigital.unicamp.br/document/?code=000429260>>. Citado na página 30.
- PICADO, J. A álgebra dos sistemas de identificação: da aritmética modular aos grupos diedrais. *Boletim da Sociedade Portuguesa de Matemática*, v. 44, p. 39–73, 2001. Disponível em: <http://www.math.ist.utl.pt/~acannas/Talentos/Apresentacoes/04_artigo_picado.pdf>. Citado na página 18.
- PINTO, E. M. et al. Sobre os Estados Emaranhados. *Physicae Organum : Revista dos Estudantes de Física da Universidade de Brasília*, v. 1, n. 1, mar. 2015. ISSN 2446-564X. Disponível em: <<http://periodicos.unb.br/index.php/physicae/article/view/13349>>. Citado 2 vezes nas páginas 7 e 46.
- PIZA, A. F. R. D. T. *Mecânica Quântica Vol. 51*. [S.l.]: EdUSP, 2003. ISBN 978-85-314-0748-2. Citado na página 22.
- SHIN, S. W. et al. How "Quantum" is the D-Wave Machine? *arXiv:1401.7087 [quant-ph]*, jan. 2014. ArXiv: 1401.7087. Disponível em: <<http://arxiv.org/abs/1401.7087>>. Citado na página 15.
- SILVA, F. L. S. D. Computação Quântica: O Algoritmo de Deutsch e o Paralelismo Quântico. *Physicae*, v. 3, n. 3, jan. 2003. ISSN 16799569, 16799569. Disponível em: <<http://www.ifi.unicamp.br/physicae/ojs-2.1.1/index.php/physicae/article/view/84>>. Citado na página 30.
- STEINBRUCH, A. *Álgebra Linear*. São Paulo (SP): Pearson, 1995. ISBN 978-0-07-450412-3. Citado 3 vezes nas páginas 20, 23 e 24.
- STRATHERN, P. *Turing e o Computador em 90 minutos*. [S.l.]: Zahar, 2000. ISBN 978-85-7110-566-9. Citado 2 vezes nas páginas 13 e 14.
- TURING, A. M. Computing Machinery and Intelligence. *Mind*, v. 59, n. 236, p. 433–460, out. 1950. ISSN 0026-4423. Disponível em: <<http://www.jstor.org/stable/2251299>>. Citado na página 47.
- ÖMER, B. *Quantum programming in QCL*. na, 2000. Disponível em: <<http://scholar.google.com/scholar?cluster=16734803813185236072&hl=en&oi=scholar>>. Citado 7 vezes nas páginas 9, 34, 38, 43, 45, 77 e 79.
- ÖMER, B. Classical Concepts in Quantum Programming. *International Journal of Theoretical Physics*, v. 44, n. 7, p. 943–955, jul. 2005. ISSN 0020-7748, 1572-9575. Disponível em: <<http://link.springer.com/10.1007/s10773-005-7071-x>>. Citado na página 74.

Apêndices

APÊNDICE A – QCL

A.1 INSTALAÇÃO DO QCL

O procedimento de instalação é direcionado a usuários que não tem familiaridade com ambiente Linux, recomenda-se ler toda essa seção para depois tentar fazer a instalação do QCL.

Alguns dos procedimentos de instalação foram retirados de [Jose, Piqueira e Lopes \(2013\)](#). E outros vem da própria experiência na instalação do compilador QCL.

O pacote para instalação da linguagem QCL está disponível no sítio <http://tph.tuwien.ac.at/oemer/qcl.html>. Lá se encontram algumas versões para instalação de um compilador QCL, mas todas elas para um sistema operacional Linux. Deve-se tomar cuidado, pois tem-se diferentes pacotes dependendo da versão de sistema operacional que se está trabalhando. O estudo realizado neste trabalho de conclusão de curso foi utilizado o sistema operacional Linux Ubuntu 14.04 de 64 bits. Os procedimentos para a instalação estão descritos a seguir:

Um usuário mais avançado não tem nenhuma dificuldade na instalação do compilador QCL, sendo que todo o procedimento deve ser realizado via terminal ou console. Após o download do pacote no referido sítio, sugere-se a criação de uma pasta somente para estes procedimentos, que deve estar em um local de fácil acesso ao usuário e não deve conter um nome muito extenso e nem vários nomes separados por espaço. Então deve-se salvar o arquivo baixado na nova pasta criada. O próximo passo é verificar se seu computador tem os compiladores necessários para a instalação do QCL. Para isso deve-se descompactar o arquivo baixado e acessar o arquivo README, nele contém um tutorial de como instalar o QCL e também os compiladores necessários, ou pelo menos a maioria deles. Esses compiladores variam de máquina para máquina.

No arquivo README, mais precisamente na seção instalação, está descrito que precisa instalar três programas: o flex, o bison e o readline. Esses três podem ser instalados pelo terminal, mas o computador deve estar conectado a internet. Basta abrir um terminal e digitar (no Ubuntu) “**sudo apt-get install *******”, onde ***** representa o nome do programa que se quer instalar. Feita a instalação pode-se prosseguir para o próximo passo. Então, pelo terminal, devemos abrir a pasta com o programa QCL salvo, e já descompactado, para isso deve-se digitar no terminal “cd caminho da pasta” como no exemplo: “cd Documentos/programas/qcl/”. Em seguida executamos no terminal o comando “**Make**”, que tem a função de montar o programa. Se nenhuma mensagem de erro aparecer podemos prosseguir digitando o comando “**Make install**”, o qual fará a

instalação do programa, podendo logo em seguida fazer uso do mesmo, lembrando que ele roda pelo terminal e para abri-lo basta digitar “**qcl**”.

Além dos mencionados compiladores e bibliotecas, deve-se ter instalado na sua máquina um compilador de C++, que no Ubuntu 14.04 pode ter sua instalação realizada com o comando “**sudo apt-get install g++**”. Se a sua máquina já tem esse compilador a sua instalação não é necessária. Talvez você não saiba se seu computador possui ou não esse compilador, mas isso é fácil de se verificar. Abra um terminal de sua preferência e digite “**g++**”, se a seguinte mensagem de erro aparecer: “**g++: fatal error: no input files compilation terminated.**”, isso significa que o seu computador já tem o compilador, qualquer outra mensagem diferente desta significa que a instalação deve ser realizada.

Se o computador deu erro por falta de algum compilador quando o comando “**Make**” foi executado, na mensagem de erro que o próprio computador dá, está descrito qual compilador está faltando, fazendo a instalação do compilador informado deve-se executar novamente o comando “**Make**”, se nenhum erro aparecer novamente é só prosseguir com a instalação. Se um novo erro for identificado, deve-se repetir o procedimento até que o comando “**Make**” compile sem erros. O processo de instalação do QCL foi realizado algumas vezes durante o desenvolvimento deste trabalho de conclusão de curso, e nota-se que duas bibliotecas sempre faltavam durante o processo, que eram a “**ncurses**” e a “**libplot**”. Ambas tem instalação simples: “**sudo apt-get install ncurses-dev**” e “**sudo apt-get install libplot-dev**”, respectivamente (comandos no terminal).

Após a instalação no ambiente Linux, mais precisamente no Ubuntu 14.04 é só abrir um terminal e digitar “**qcl**”, como o mencionado. Se tudo ocorreu bem durante a instalação do programa deve aparecer algo parecido com o que está ilustrado abaixo:

```
nome-usuario@nomeusuario-C14RV01: ~$qcl
QCL Quantum Computation Language (64 qubits, seed 1442426701)
[0/64]1|0 >
qcl>
```

Após todos esses passos o compilador QCL está pronto para o uso (ÖMER, 2005).

Anexos

ANEXO A – TABELAS DE OPERADORES

QCL

Tabela 6 – Tabela de Operadores com base na tabela de [Ömer \(2000\)](#)

Operador	Descrição	Tipo de argumento
#	Tamanho registro	Tipo quântico
^	Potência e Potência inteira	Todos os aritméticos, int
-	Menos Unitário	Todos os aritméticos
*	Multiplicação	Todos os aritméticos
/	Divisão e divisão por inteiro	Todos os aritméticos, int
mod	Módulo Inteiro	int
+	Adição	Todos os aritméticos
-	Subtração	Todos os aritméticos
&	Concatenação	Tipo quântico
==	Igual	Todos os aritméticos
!=	Diferente	Todos os aritméticos
<	Maior	Inteiro, Real
<=	Maior ou ingual	int, real
>	Menor	int, real
>=	Menor ou igual	int, real
not	Não lógico	Booleana
and	e Lógico	Booleana
or	Lógica inclusiva para	Booleana
xor	Lógica exclusiva para	Booleana

ANEXO B – TABELA DE FUNÇÕES

Tabela 7 – Funções definidas em QCL com base nas tabelas de Ömer (2000)

Funções Trigonométricas		Funções Hiperbólicas	
$\sin(x)$	Seno de x	$\sinh(x)$	Seno hiperbólico de x
$\cos(x)$	Coseno de x	$\cosh(x)$	Coseno hiperbólico de x
$\tan(x)$	Tangente de x	$\tanh(x)$	Tangente hiperbólico de x
$\cot(x)$	Cotangente de x	$\coth(x)$	Cotangente Hiperbólico de x
Funções Complexas		Exponencial e Funções Relacionadas	
$Re(z)$	Parte real de z	$\exp(x)$	Exponencial de x
$Im(z)$	Parte imaginária de z	$\log(x)$	Logarítimo natural de x
$abs(z)$	Magnetude de z	$\log(x, n)$	Logarítimo de x na base n
$conj(z)$	Conjugado de z	\sqrt{x}	Raiz quadrada de x

ANEXO C – TABELA DE EQUIVALÊNCIA ENTRE DECIMAIS, BINÁRIO E HEXADECIMAL

Tabela 8 – Equivalência entre escalas

Decimal	Binário	Hexadecimal
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7
8	1000	8
9	1001	9
10	1010	A
11	1011	B
12	1100	C
13	1101	D
14	1110	E
15	1111	F